WRF Computing Best Practices

Adding or modifying compilable WRF source code

Adding or modifying compilable WRF source code

- Does WRF care about free *vs* fixed form?

- The WRF source code is mostly a free-formatted Fortran code base.
- The configure.wrf file has explicit rules for the free and fixed codes.
- The WRF build system assumes that all of user contributed code will compile as free-formatted.

Adding or modifying compilable WRF source code

- Is there a Fortran standard to which the user should adhere?

- There are a few Fortran 2003 capabilities included, and most recent compilers support these.
- Do not attempt to utilize coarrays or other more exotic additions to the Fortran standard.
- Compilers vary in there support for newer capabilities, both being able to compile them, and to use them efficiently.

Adding or modifying compilable WRF source code

- How should the call to my new physics package be done?

- Look in the specific driver for examples of existing calls. For example phys/ module_radiation_driver.F

Adding or modifying compilable WRF source code

- How should the call to my new physics package be done?

- All top level physics routines are called with a 3d block of data.
- 18 dimensions are always passed:
- I, J, K dimensions
- domain, memory, and computational sizes
- starting and ending

Adding or modifying compilable WRF source code

- How should the call to my new physics package be done?

```
CALL cal_cldfra1(CLDFRA,qv,qc,qi,qs,                &
                 F_QV,F_QC,F_QI,F_QS,t,p,           &
                 F_ICE_PHY,F_RAIN_PHY,              &
                 ids,ide, jds,jde, kds,kde,         &
                 ims,ime, jms,jme, kms,kme,         &
                 its,ite, jts,jte, kts,kte          )
```

Adding or modifying compilable WRF source code

- Where does WRF assume that the values inside of the physics schemes are located?

- The physics schemes are column oriented, no communications are required top to bottom.
- The values are located at mass points.
- Some variables are located on full eta level (usually denoted with the cryptic convention “8w”), but most 3d variables are located on the computational half layer locations.

Adding or modifying compilable WRF source code

- If a new variable was added in the Registry, at what point does this get manually introduced in the subroutine calling tree?

- All variables in the Registry (state + namelist options) are in the derived data structure “grid”.
- When “grid” is available, the new variable does not need to be dereferenced from the structure.
- The calls to the drivers in module_first_rk_step_part1.F exhibit the required dereferencing.

Adding or modifying compilable WRF source code

- If a new variable was added in the Registry, at what point does this get manually introduced in the subroutine calling tree?

- **CALL radiation_driver(&**
- **ACFRCV=grid%acfrcv , &**
- **ACFRST=grid%acfrst , &**
- **ALBEDO=grid%albedo , &**

Adding or modifying compilable WRF source code

- If a new variable was added in the Registry, at what point does this get manually introduced in the subroutine calling tree?

- The call to the specific driver needs to have the new variable explicitly passed in from module_first_rk_step_part1.F (or from the solver for the microphysics routines).
- User modifications are then required in all deeper routines.

Adding or modifying compilable WRF source code

- If there is a new variable that needs to be communicated, how is that set up in WRF?

- All communications in WRF are a combination of two items: manual inclusion of compilable source code, and manual inclusion of communications information in the Registry.
- The source modifications “cpp include” a file into the source prior to compilation.

Adding or modifying compilable WRF source code

- If there is a new variable that needs to be communicated, how is that set up in WRF?

- The developer may choose to communicate the variables immediately after the computation is performed to manufacture the new variable, or wait until the new variable's halo is needed.

Adding or modifying compilable WRF source code

- If there is a new variable that needs to be communicated, how is that set up in WRF?

- **#ifdef DM_PARALLEL**
- **# include "PERIOD_BDY_EM_A.inc"**
- **#endif**

Adding or modifying compilable WRF source code

- How do you access a particular hydrometeor from the 4d array moist?

- The name associated with the variable defined in the Registry is used to construct a Fortran PARAMETER value.
- This integer index should always be used to refer to the particular 3d array.

Adding or modifying compilable WRF source code

- How do you access a particular hydrometeor from the 4d array moist?

- Registry, first few parts of the QVAPOR line:
- **`state real qv ikjftb moist`**

- Source code:
- **`qvf = 1. + rvovrd*moist(i,k,j,P_QV)`**

Adding or modifying compilable WRF source code

- How do you access a particular hydrometeor from the 4d array moist?

- Loops over the 4d arrays should always begin and end with the WRF specific starting values:

- **DO im = PARAM_FIRST_SCALAR, num_3d_m**
- **qtot = qtot + moist(i,k,j,im)**
- **ENDDO**

Adding or modifying compilable WRF source code

- How do you access a particular hydrometeor from the 4d array moist?

- These automatically generated values are inside module_state_description.
- When these generated indexes are required for new code, USE module_state_description.

Adding or modifying compilable WRF source code

- With modern Fortran, how do I get information from the module?

- A “use association” is employed in WRF.
- To restrict the number of symbol names that are shared, the WRF code tends to restrict the variables requested with the ONLY clause.
- Mostly this “ONLY clause” is added to keep compilers from complaining about source code being “too complex” when the used module is large.

Adding or modifying compilable WRF source code

- With modern Fortran, how do I get information from the module?

- `USE module_configure, ONLY : grid_config_rec_type`
- `USE module_driver_constants`
- `USE module_machine`
- `USE module_tiles, ONLY : set_tiles`

Adding or modifying compilable WRF source code

- With modern Fortran, how do I get information from the module?

- Typically when adding in communications or new physics packages, the USE statements need to be amended to include the new Registry information.

Adding or modifying compilable WRF source code

- If there are known restrictions for packages, how can that information be used at model initialization?

- There are two mechanisms in WRF for handling error checking for the physics schemes:
- phys/module_physics_init.F
- share/module_check_a_mundo.F

Adding or modifying compilable WRF source code

- If there are known restrictions for packages, how can that information be used at model initialization?

- The tests in physics_init are more aligned with modifying 2d and 3d arrays depending on the values of namelist settings or other 2d and 3d arrays. Initializations for each domain take place.
- To avoid OpenMP race conditions, this is a much better way to fix zeroed-out variables.

Adding or modifying compilable WRF source code

- If there are known restrictions for packages, how can that information be used at model initialization?

- The purpose for check_a_mundo is to stop incompatible namelist options. If a user knows that a certain scheme is only set up to work with one type of PBL, then that needs to be included.

Adding or modifying compilable WRF source code

- What does the WRF model mean by “restart”, and how does it impact a developer?

- A restart in WRF allows a model simulation to continue from an interrupted state, AND to produce bit-wise identical results to those generated from a non-interrupted simulation.
- Developers need to provide information as to which variables need to be saved for a restart run.

Adding or modifying compilable WRF source code

- What does the WRF model mean by “restart”, and how does it impact a developer?

- The restart variables are explicitly listed in the Registry.

- **state real rimi ikj misc **
- **1 - irh **
- **"RIMI" "riming intensity" \ "fraction"**

Adding or modifying compilable WRF source code

- What does the WRF model mean by “restart”, and how does it impact a developer?

- The physics_init routine needs to avoid resetting restarted variables, which requires user modification.
- **`IF(.not.restart)THEN`**
- **`IF(PRESENT(rliq)) THEN`**
- **`    rliq(:,:) = 0.0`**
- **`  ENDIF`**
- **`ENDIF`**

Adding or modifying compilable WRF source code

- Will the WRF community sing my praises if I, as a developer, include lots of inline documentation?

- **YES!**

Adding or modifying compilable WRF source code

- Is there a mechanism to output the namelist options that are being developed?

- The file share/output_wrf.F handles the metadata output:

```
• ibuf(1) = config_flags%e_we - config_flags%s_we + 1
• CALL wrf_put_dom_ti_integer ( fid , &
•    'WEST-EAST_GRID_DIMENSION' ,  ibuf , 1 , ierr )
```

Adding or modifying compilable WRF source code

- Is there a mechanism to output the namelist options that are being developed?

- Routines exist to output integer, real, logical, and character strings.

Adding or modifying compilable WRF source code

- What WRF infrastructure exists to make coding easier for such processing as global sums, global extrema and locations of extrema?

- There are a few routines in WRF that handle these types of capabilities for most traditional data types (real, double, integer).

Adding or modifying compilable WRF source code

- What WRF infrastructure exists to make coding easier for such processing as global sums, global extrema and locations of extrema?

- There are a few routines in WRF that handle these types of capabilities for most traditional data types (real, double, integer).

Adding or modifying compilable WRF source code

- What WRF infrastructure exists to make coding easier for such processing as global sums, global extrema and locations of extrema?

- **`lat1 = wrf_dm_min_real ( lat1 )`**

- Please see example #3 for a more complete list and examples of usage:
- http://www2.mmm.ucar.edu/wrf/users/tutorial/201401/WRF_Registry_2.pdf

Adding or modifying compilable WRF source code

- How about initializations that can only be handled on the master node?

- Even for serially-built code, the following is defined:

- **`IF ( wrf_dm_on_monitor() ) THEN`**

Adding or modifying compilable WRF source code

- How does info get from the master node to the other processors?

- Again for native data types, a variable (and the number of words) can be broadcast to all of the processors from the master.

- **`CALL wrf_dm_bcast_integer(nt,1)`**

Adding or modifying compilable WRF source code

- If information is in the namelist, how is it accessed inside the code?

- There are three methods to get namelist information:
- grid%
- config_flags%
- subroutine calls

Adding or modifying compilable WRF source code

- If information is in the namelist, how is it accessed inside the code?

- Any time the “grid” structure is present, the namelist option may be dereferenced as an existing field in the derived structure:

- `p_top_requested = grid%p_top_requested`

Adding or modifying compilable WRF source code

- If information is in the namelist, how is it accessed inside the code?

- Similarly, the derived structure config_flags holds the namelist information for the current grid being processed:

```
IF ( config_flags%spec_bdy_width .GT. &
     flag_excluded_middle ) THEN
```

Adding or modifying compilable WRF source code

- If information is in the namelist, how is it accessed inside the code?

- The WRF code automatically builds two subroutines for each namelist variable, a “get” and a “set” subroutine. Mostly, developers are interested in the “get” option. Argument #1 is which domain, and argument #2 is the local returned value.

- **`CALL nl_get_base_pres  ( 1 , p00 )`**

Adding or modifying compilable WRF source code

- What are the available options for outputting debug print information?

- Because not all print buffers are guaranteed to be flushed on an error exit, it is better to use WRF supplied print-out functions.
- Also, use the WRF provided fatal error function instead of a Fortran STOP statement.

Adding or modifying compilable WRF source code

- What are the available options for outputting debug print information?

- `CALL wrf_debug ( 200 , ' call end of solve_em' )`
- `CALL wrf_message('ndown: using namelist constants')`
- `CALL wrf_error_fatal( 'Use km_opt=2 with sfs_opt=2')`

Adding or modifying compilable WRF source code

- If a developer wants an event to occur every so often, how is that accomplished?

- Be wary of a simple
- MOD (current_time , some_interval) == 0
- set up. For large values of current time, the statement may eventually never be true again. For a fixed time step, the integer number of time steps might be preferable:
- **`IF (mod(itimestep,STEPFG) .eq. 0) THEN`**

Adding or modifying compilable WRF source code

- What is supposed to happen with OPTIONAL variables and the CPP ifdef'ing?

- First, this is required due to the two different dynamical cores inside of WRF, and even for ARW the DA and Chem codes do not need all variables. To allow the physics schemes to work with both cores (and the Chem and DA options), some variables are considered optional because they are not present at all times.

Adding or modifying compilable WRF source code

- What is supposed to happen with OPTIONAL variables and the CPP ifdef'ing?

- For a new scheme, if only a "few" variables are to added to both cores, it is reasonable to add the variables to both the ARW and NMM Registry files (similarly, the DA and Chem Registry files).
- If LOTS of variables are to be added, it is better to go the OPTIONAL variable route.

Adding or modifying compilable WRF source code

- What is supposed to happen with OPTIONAL variables and the CPP ifdef'ing?

- The variables that are only required for one of the build options are ifdef'ed out in the calling routine (for example the first_rk_step_part1 file
- **`#ifdef WRF_CHEM`**
- **`  & ,CHEM=chem,chem_opt=config_flags%chem_opt`**
- **`#endif`**

Adding or modifying compilable WRF source code

- What is supposed to happen with OPTIONAL variables and the CPP ifdef'ing?

- Always add new variables to a Registry package to minimize the model's memory footprint. The variables are allocated, but only with a (1,1,1) size.
- When using OPTIONAL arguments, always test if the variable is PRESENT before using.

Adding or modifying compilable WRF source code

- What is supposed to happen with OPTIONAL variables and the CPP ifdef'ing?

```
IF ( PRESENT( rainshv )) THEN
   DO j=j_start(ij),j_end(ij)
      DO i=i_start(ij),i_end(ij)
         RAIN(i,j) = RAIN(i,j) + RAINSHV(i,j)
      ENDDO
   ENDDO
END IF
```

Adding or modifying compilable WRF source code

- What are the usual ifdef syntaxes that are to be used?

- To avoid the situation where a compile-time option is set to zero (where the intent was to turn the option OFF), the WRF ifdef's test on the number "1".

Adding or modifying compilable WRF source code

- What are the **#ifdef** syntaxes that are to be used?

- **#if (DA_CORE == 1)**

- **#if (WRF_CHEM == 1)**

- **#if (NMM_CORE == 1)**

- **#if (EM_CORE == 1)**

Adding or modifying compilable WRF source code

- What type of communications are allowed between columns in the physics schemes?

- By default, the physics schemes are column oriented, with no impact permitted from neighbors.
- This means NO horizontal differences or horizontal averaging inside the physics schemes.

Adding to or modifying the WRF Registry

Adding to or modifying the WRF Registry

- What is the WRF Registry?

- The WRF Registry is an active data dictionary.
- It is a text-based file that is user-modifiable.
- Every variable used with I/O, communications, namelist option is in the Registry.
- All associations of variables with physics schemes are handled by the Registry.

Adding to or modifying the WRF Registry

- What is the WRF Registry?

- The text-based file is read by a program.
- This registry program manufactures include files that are CPP #include'd during the WRF build process.
- More than 300 thousand lines of automatically generated code are included in the WRF source code via the registry program.

Adding to or modifying the WRF Registry

- What are the different types of model variables in the Registry?

- Most users are concerned with the gridded data or with the namelist variables. The Registry handles both of these.
- The gridded data is either “state” (available throughout the duration of the simulation) or the data is “i1” (tendency variables that pop off the stack at the conclusion of each time step).

Adding to or modifying the WRF Registry

- What are the different types of model variables in the Registry?

- Please see for more information on the Registry:

- **www.mmm.ucar.edu/wrf/users/tutorial/201401/WRF_Registry_1.pdf**

Adding to or modifying the WRF Registry

- How is I/O handled in the Registry?

- There are multiple streams (think of these as separate unit numbers) for input and output.
- Each variable may be in zero or more streams.
- The WRF naming convention for the streams:
- i => input
- h => history
- r => restart

Adding to or modifying the WRF Registry

- How is I/O handled in the Registry?

```
#<Table> <Type> <Sym> <Dims>
state     real   LAT    ij

#<Use>   <NumTLev> <Stagger>
misc        1          -

#           <IO>
i0123rh01{22}{23}du=(copy_fcnm)

#<DNAME>              <DESCRIP>                      <UNITS>
 "XLAT"     "LATITUDE, SOUTH IS NEGATIVE"  "degree_north"
```

Adding to or modifying the WRF Registry

- How is I/O handled in the Registry?

- For input and history, the default stream number is “0”.
- The default input stream: wrfinput_*<domain>*
- The default output stream: wrfout_*<domain>*_*<date>*

Adding to or modifying the WRF Registry

- How is I/O handled in the Registry?

- A stream specification of “ih” assumes the field is in the input stream and will be output to the WRF history file. The numeral zero is assumed present if there are no numerals.
- Numerals are added after the characters “i” or “h” to indicate additional (nonstandard) streams for the fields.

Adding to or modifying the WRF Registry

- How is I/O handled in the Registry?
- Once explicit stream numbers are specified, the “zero” stream must also be specifically requested, as in i01.
- Streams with more than one digit, for example stream #14, would be surrounded by “{}”, as in {14}

Adding to or modifying the WRF Registry

- How is I/O handled in the Registry?

- First couple entries for the eta levels:
- **`state real znu  k  dyn_em 1 - irh`**
- **`state real znw  k  dyn_em 1 Z i0rh`**

- Note that irh and i0rh are identical specifications.

Adding to or modifying the WRF Registry

- How is I/O handled in the Registry?

- The 2-m temperature is available for input from real (i0), input from metgrid (i1), output to the default history file (h0), and output to an auxiliary stream (h{23}):

- **`state real T2    ij misc   1 - i01rh0{23}`**

Adding to or modifying the WRF Registry

- How is I/O handled in the Registry?

- All variables involved with I/O are required to be state.
- The state variables may be real, double, integer, character, or logical.
- Variables for I/O must be 0d, 1d, 2d, 3d, or part of a known 4d amalgamation.
- Only one time slice of two-time-level fields is output.

Adding to or modifying the WRF Registry

- How is I/O handled in the Registry?

- 2d arrays must be (i,j).
- 3d arrays must be decomposed in (i,j).
- 4d arrays must be the special scalar-type aggregations of 3d arrays (they are processed as lists of 3d array elements).

Adding to or modifying the WRF Registry

- How is I/O handled in the Registry?

- The Registry is not involved in the actual format of the input or output data.
- The format, frequency, name of the file, *etc.* are all run-time options (though the namelist options controlling those capabilities are defined in the Registry).

Adding to or modifying the WRF Registry

- How is I/O handled in the Registry?

- Only use an “i” for variables that are input. For example, convective precipitation is not input from the real program, and should not have an “i”.
- Similarly, developers tend to think every variable that was used is vital. Judiciously select those that will be given an “h” designation.

Adding to or modifying the WRF Registry

- How is I/O handled in the Registry?

- The “r” designator is mandatory for fields that are required to manufacture an identical simulation, when comparing a restart run to a non-restart run.

- Including an “r” for non-mandatory fields makes the restart file larger and the associated I/O slower, but otherwise has no negative forecast impact.

Adding to or modifying the WRF Registry

- How is nesting handled in the Registry?

- The same block of information controlling the I/O has a few keywords that control the nesting.
- u => feedback up to parent mesh
- d => horizontally interpolate down to child domain
- f => lateral boundary forcing
- s => smoothing on CG in area of FG

Adding to or modifying the WRF Registry

- How is nesting handled in the Registry?

- The “u”, “d”, and “f” options are able to use a default for most continuous variables (though the horizontal staggering is important).
- Developers may associate a new subroutine with a new physics variable, though this is not too common.
- Almost all of the lateral boundary forcing is the dynamics variables, with no usage for the physics variables.

Adding to or modifying the WRF Registry

- How is nesting handled in the Registry?

- As with most Registry items, it is usually safest for a developer to copy a similar (and existing) Registry line for the initial idea for a new variable.

Adding to or modifying the WRF Registry

- How is nesting handled in the Registry?

- Developers handling land surface fields must be concerned with masking.
- An average across the spatial extent of a parent cell might include both water and land points from the child, which would feedback garbage to the parent.

Adding to or modifying the WRF Registry

- How is nesting handled in the Registry?

- While the mnemonics of “u” and “d” refer to “up” and “down”, respectively, the WRF nesting code is general.
- d => once only, at the start of the model simulation
- u => child to parent, at the end of each set of child time steps (that bring the child and parent to the same time)
- f => parent to child, at the start of each set of child time steps

Adding to or modifying the WRF Registry

- How is nesting handled in the Registry?

- For example, CG SST may be handed to the FG at each CG step via an “f” option (subroutine: c2f_interp):
- `state   real     OM_TMP \`
- `i{nocnl}j       misc      1     Z \`
- `i012rhdu=(copy_fcnm) \`
- `f=(c2f_interp:grid_id)     \`
- `"OM_TMP" "temperature"    "k"`

Adding to or modifying the WRF Registry

- How is communication handled in the Registry?

- There are three kinds of communications possible with WRF:
- halo => next door neighbor
- period => west-east or south-north exchange
- transpose => inside ARW proper - largely for FFTs
- Most developers (if they are concerned) are only concerned with halo communications.

Adding to or modifying the WRF Registry

- How is communication handled in the Registry?
- The halo comms are specified for a list of variables, and the size of the stencil for each those variables.
- **`halo HALO_EM_PHYS_A  dyn_em 4:u_2,v_2`**
- Please see for more information on WRF stencils:
- **`www.mmm.ucar.edu/wrf/users/tutorial/201401/WRF_Software.pdf`**

Adding to or modifying the WRF Registry

- How is communication handled in the Registry?

- Overspecifying the size of the stencil has no ill effects on results, it is just a performance sink.
- The same communication pattern may “used” inside of WRF multiple times.

```
#ifdef DM_PARALLEL
#     include "HALO_EM_PHYS_A.inc"
#endif
```

Adding to or modifying the WRF Registry

- How does the Registry help with memory management?

- The Registry offers a “package” option which associates state variables with particular namelist options.
- Developers should include this for their schemes.
- Non-used package variables are allocated only with 1 word of space (for example: (1,1,1) for a 3d array, and (1,1) for a 2d array).

Adding to or modifying the WRF Registry

- How does the Registry help with memory management?

- The package option is able to handle conditional namelist settings through the use of derived namelist settings.
- The data used from metgrid by the real program is not required by the WRF model, so it is in a package controlled by a derived namelist variable.

Adding to or modifying the WRF Registry

- How does the Registry help with memory management?

- rconfig integer use_wps_input \
- derived 1 0

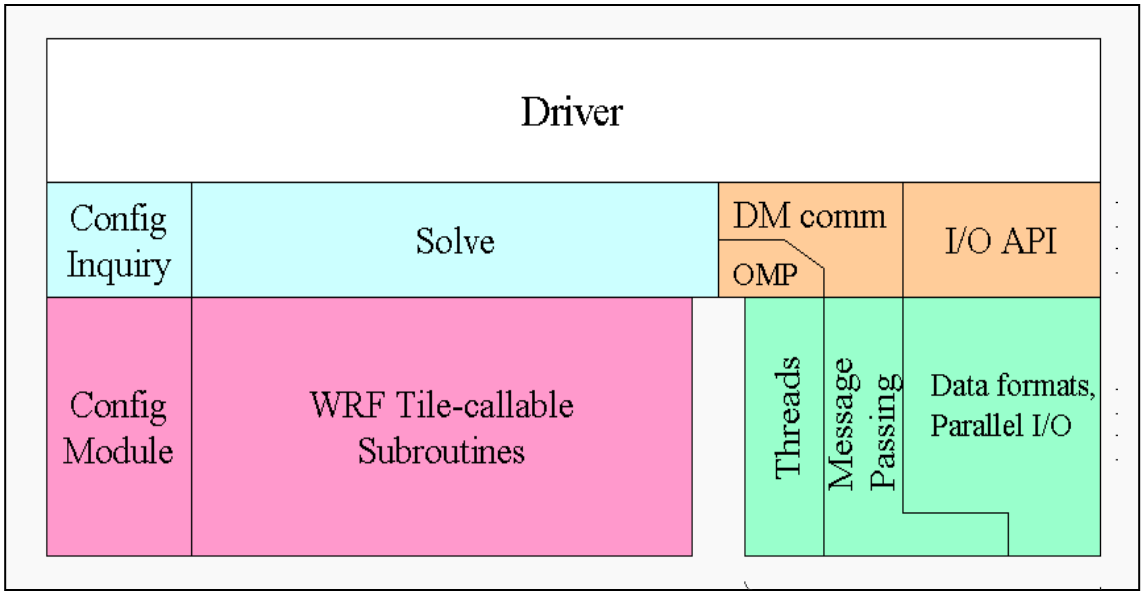
- package realonly use_wps_input==1 - \
  state:u_gc,v_gc,…

Talk #1: WRF Parallelism Best Practices

- Review of WRF: patch, tile, halo
- How MPI and OpenMP fit into WRF parallelism
- Strong vs weak scaling
- Choosing domains: appropriate to core count, aspect ratio, timing vs memory
- Domain and computation decomposition: MPI and OpenMP
- Impact of nesting on core counts
- Nesting overhead

Patch, Tile, Halo

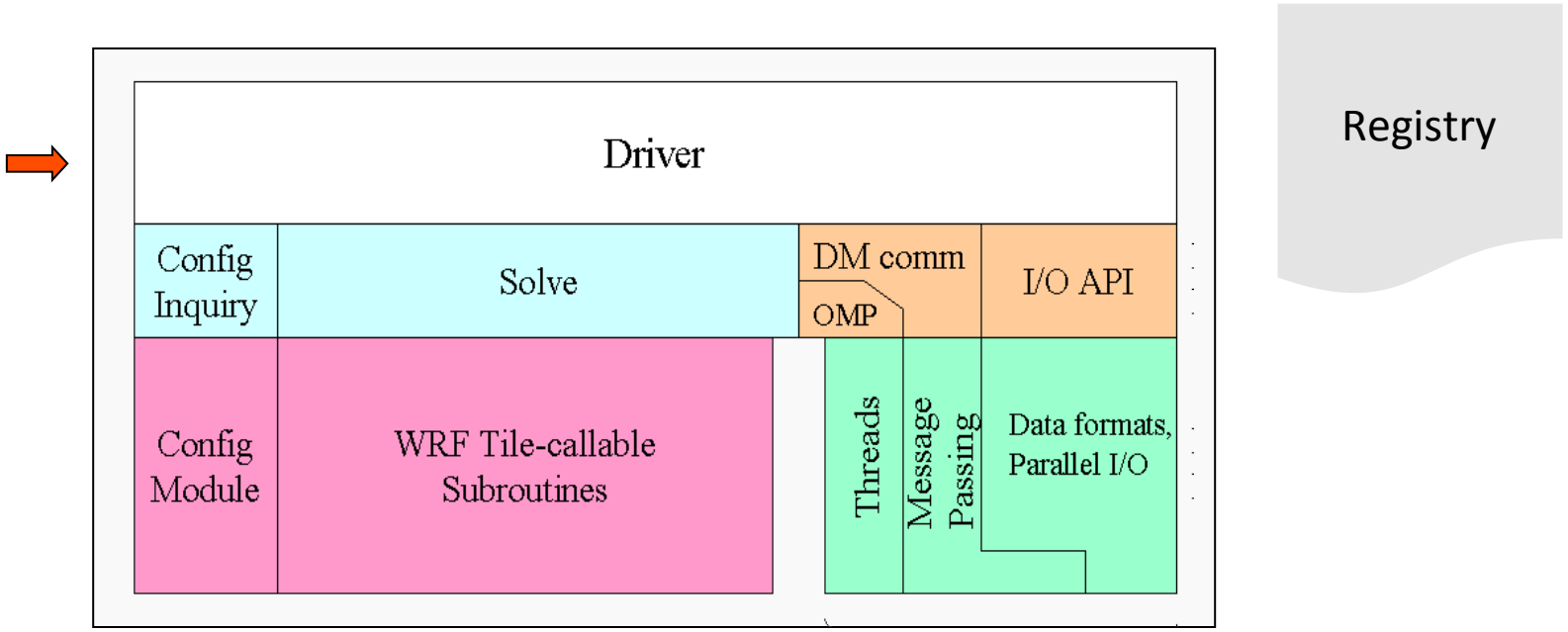
WRF Software Architecture



Registry

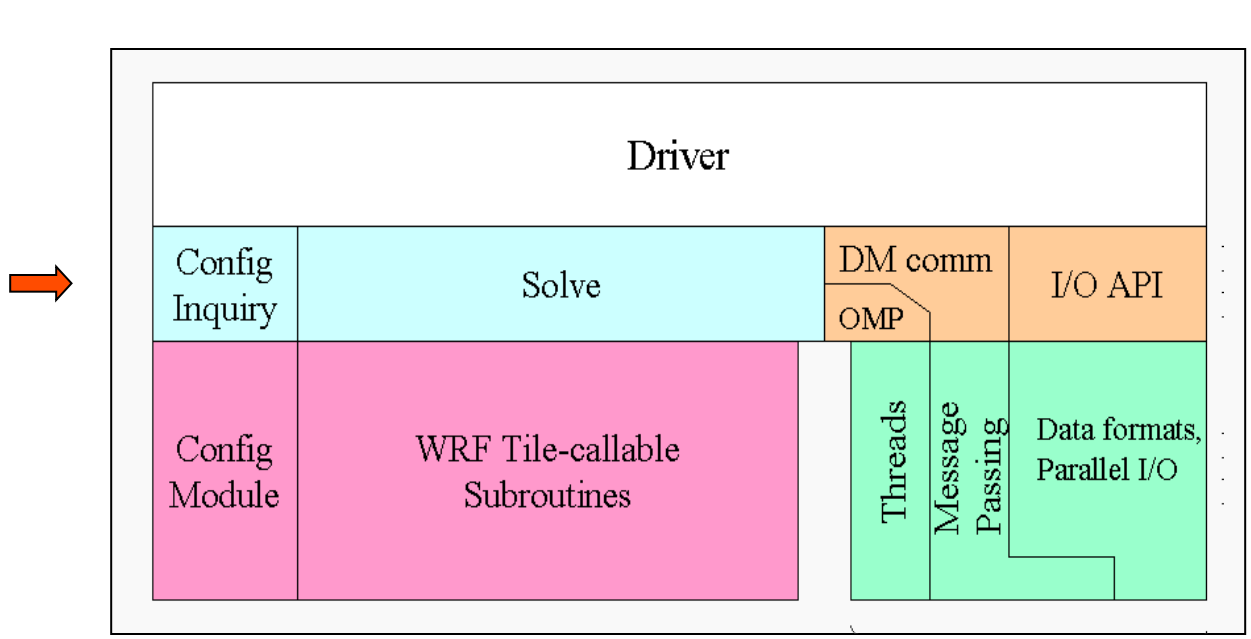
- Hierarchical software architecture
  - Insulate scientists' code from parallelism and other architecture/implementation-specific details
  - Well-defined interfaces between layers, and external packages for communications, I/O, and model coupling facilitates code reuse and exploiting of community infrastructure, e.g. ESMF.

WRF Software Architecture



- Driver Layer
  - **Domains**: Allocates, stores, decomposes, represents abstractly as single data objects
  - **Time loop**: top level, algorithms for integration over nest hierarchy

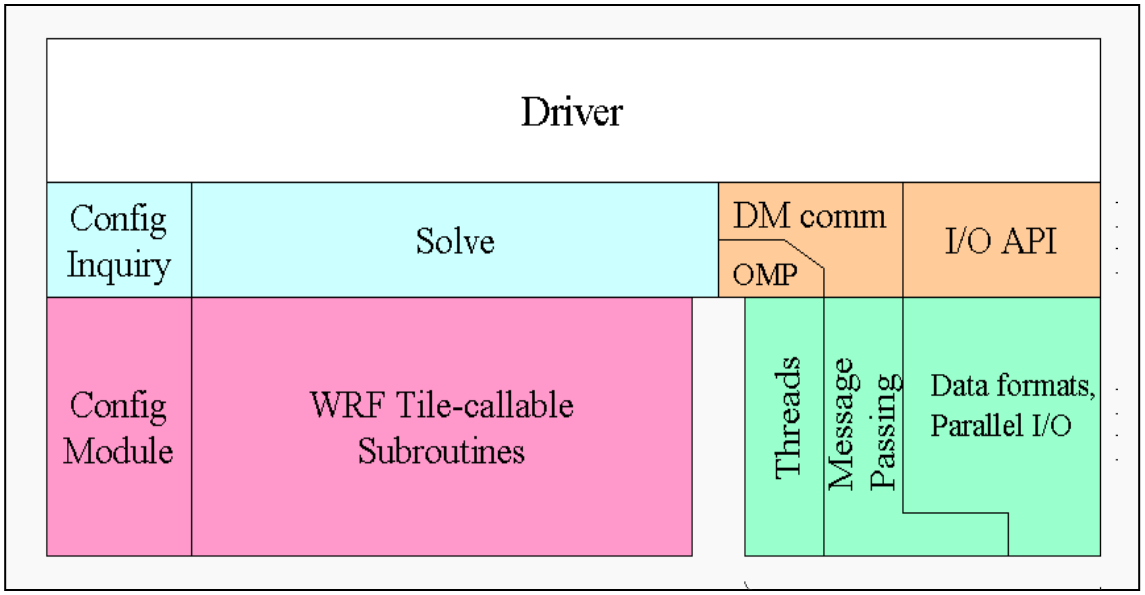
WRF Software Architecture



Registry

- Mediation Layer
  - Solve routine, takes a domain object and advances it one time step
  - Nest forcing, interpolation, and feedback routines

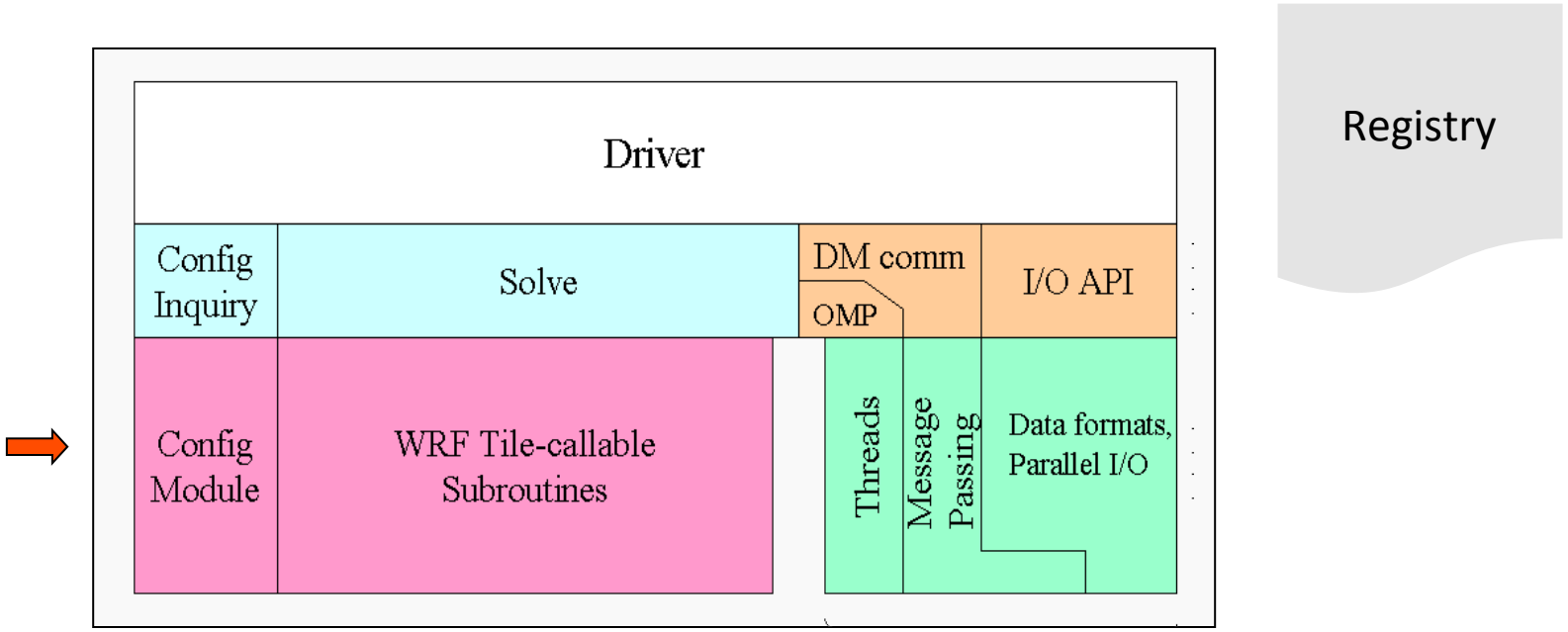
WRF Software Architecture



Registry

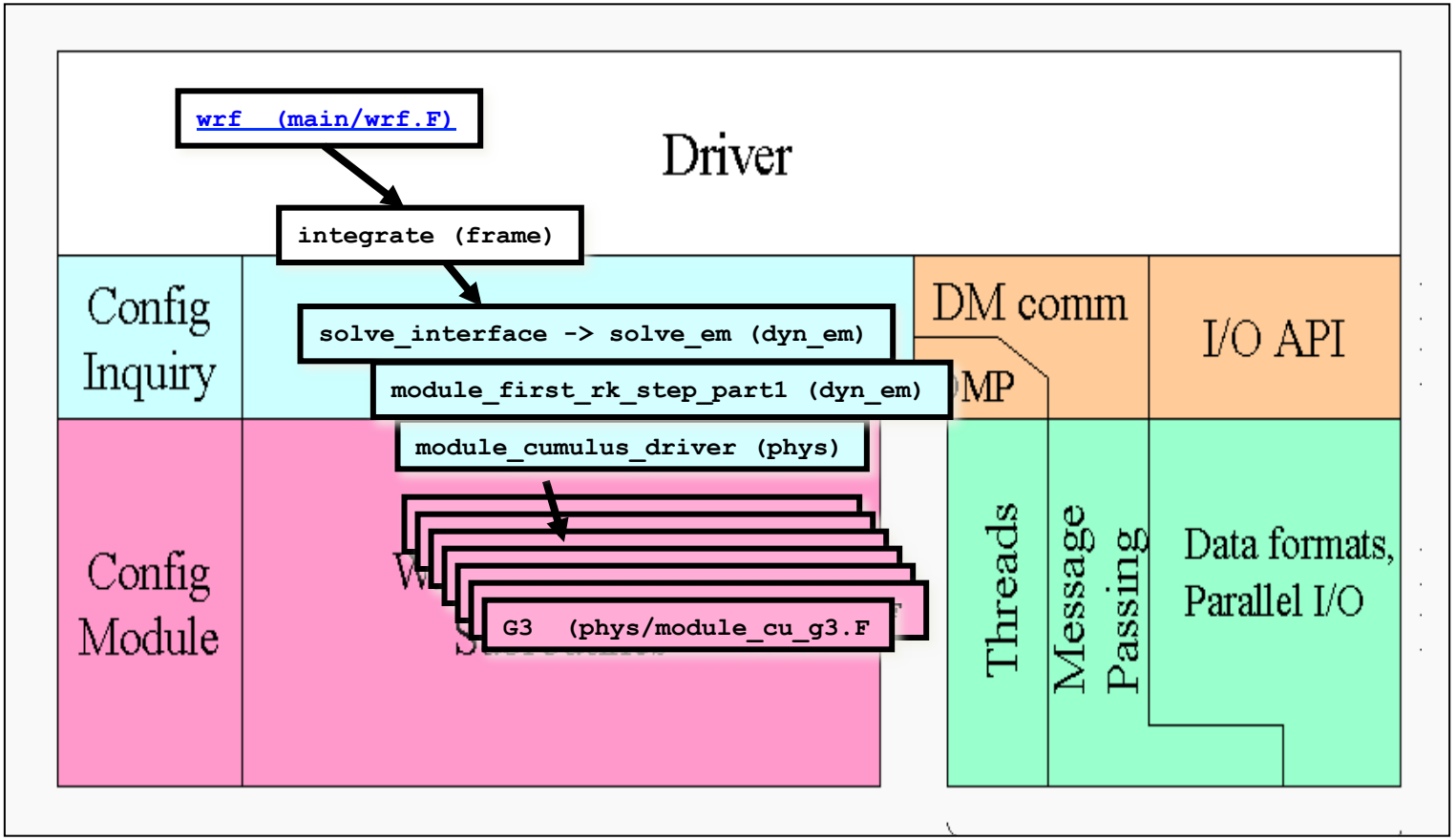
- Mediation Layer
  - The sequence of calls for doing a time-step for one domain is known in Solve routine
  - Dereferences fields in calls to physics drivers and dynamics code
  - Calls to message-passing are contained here as part of Solve routine

WRF Software Architecture



- Model Layer
  - **Physics and Dynamics**: contains the actual WRF model routines are written to perform some computation over an arbitrarily sized/ shaped, 3d, rectangular subdomain

Call Structure Superimposed on Architecture



Distributed Memory Communications

When Needed?

Communication is required between patches when a horizontal index is incremented or decremented on the right-hand-side of an assignment.

Why?

On a patch boundary, the index may refer to a value that is on a different patch.

Following is an example code fragment that requires communication between patches

Signs in code

Note the tell-tale +1 and –1 expressions in indices for **rr**, **H1,** and **H2** arrays on right-hand side of assignment.

These are ***horizontal data dependencies*** because the indexed operands may lie in the patch of a neighboring processor. That neighbor's updates to that element of the array won't be seen on this processor.

Distributed Memory Communications

(module_diffusion.F)

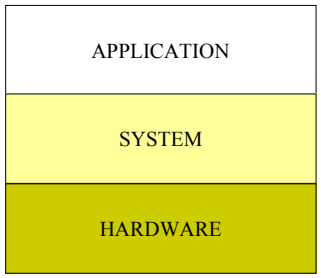
```
SUBROUTINE horizontal_diffusion_s (tendency, rr, var, . . .
. . .
   DO j = jts,jte
   DO k = kts,ktf
   DO i = its,ite
      mrdx=msft(i,j)*rdx
      mrdy=msft(i,j)*rdy
      tendency(i,k,j)=tendency(i,k,j)-                                  &
           (mrdx*0.5*((rr(i+1,k,j)+rr(i,k,j))*H1(i+1,k,j)-              &
                      (rr(i-1,k,j)+rr(i,k,j))*H1(i  ,k,j))+             &
            mrdy*0.5*((rr(i,k,j+1)+rr(i,k,j))*H2(i,k,j+1)-              &
                      (rr(i,k,j-1)+rr(i,k,j))*H2(i,k,j  ))-             &
            msft(i,j)*(H1avg(i,k+1,j)-H1avg(i,k,j)+                     &
                       H2avg(i,k+1,j)-H2avg(i,k,j)                      &
                                )/dzetaw(k)                             &
           )
   ENDDO
   ENDDO
   ENDDO
 . . .
```

Distributed Memory Communications

```
                    (module_diffusion.F )

SUBROUTINE horizontal_diffusion_s (tendency, rr, var, . . .
. . .
   DO j = jts,jte
   DO k = kts,ktf
   DO i = its,ite
      mrdx=msft(i,j)*rdx
      mrdy=msft(i,j)*rdy
      tendency(i,k,j)=tendency(i,k,j)-                                  &
           (mrdx*0.5*((rr(i+1,k,j)+rr(i,k,j))*H1(i+1,k,j)-              &
                      (rr(i-1,k,j)+rr(i,k,j))*H1(i   ,k,j))+            &
            mrdy*0.5*((rr(i,k,j+1)+rr(i,k,j))*H2(i,k,j+1)-              &
                      (rr(i,k,j-1)+rr(i,k,j))*H2(i,k,j   ))-            &
            msft(i,j)*(H1avg(i,k+1,j)-H1avg(i,k,j)+                     &
                       H2avg(i,k+1,j)-H2avg(i,k,j)                      &
                                )/dzetaw(k)                             &
           )
   ENDDO
   ENDDO
   ENDDO
 . . .
```

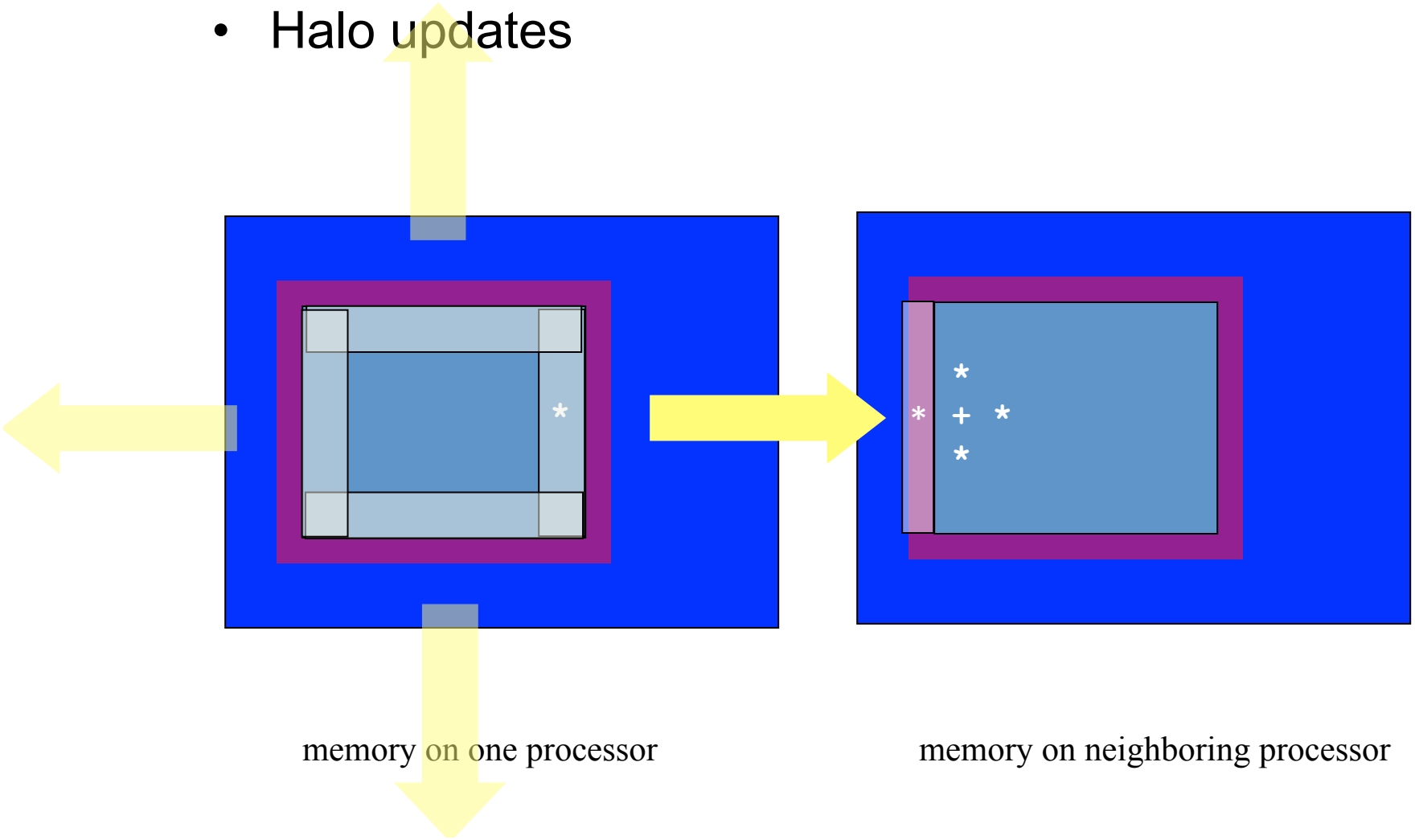
Distributed Memory Communications

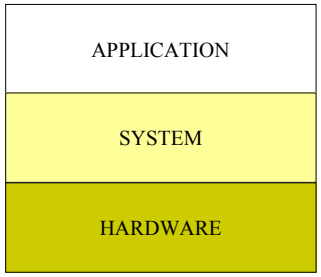
```
                    (module_diffusion.F )

SUBROUTINE horizontal_diffusion_s (tendency, rr, var, . . .
. . .
   DO j = jts,jte
   DO k = kts,ktf
   DO i = its,ite
      mrdx=msft(i,j)*rdx
      mrdy=msft(i,j)*rdy
      tendency(i,k,j)=tendency(i,k,j)-                              &
           (mrdx*0.5*((rr(i+1,k,j)+rr(i,k,j))*H1(i+1,k,j)-          &
                      (rr(i-1,k,j)+rr(i,k,j))*H1(i  ,k,j))+         &
            mrdy*0.5*((rr(i,k,j+1)+rr(i,k,j))*H2(i,k,j+1)-          &
                      (rr(i,k,j-1)+rr(i,k,j))*H2(i,k,j  ))-         &
            msft(i,j)*(H1avg(i,k+1,j)-H1avg(i,k,j)+                 &
                       H2avg(i,k+1,j)-H2avg(i,k,j)                  &
                                )/dzetaw(k)                         &
           )
   ENDDO
   ENDDO
   ENDDO
 . . .
```



Distributed Memory MPI Communications

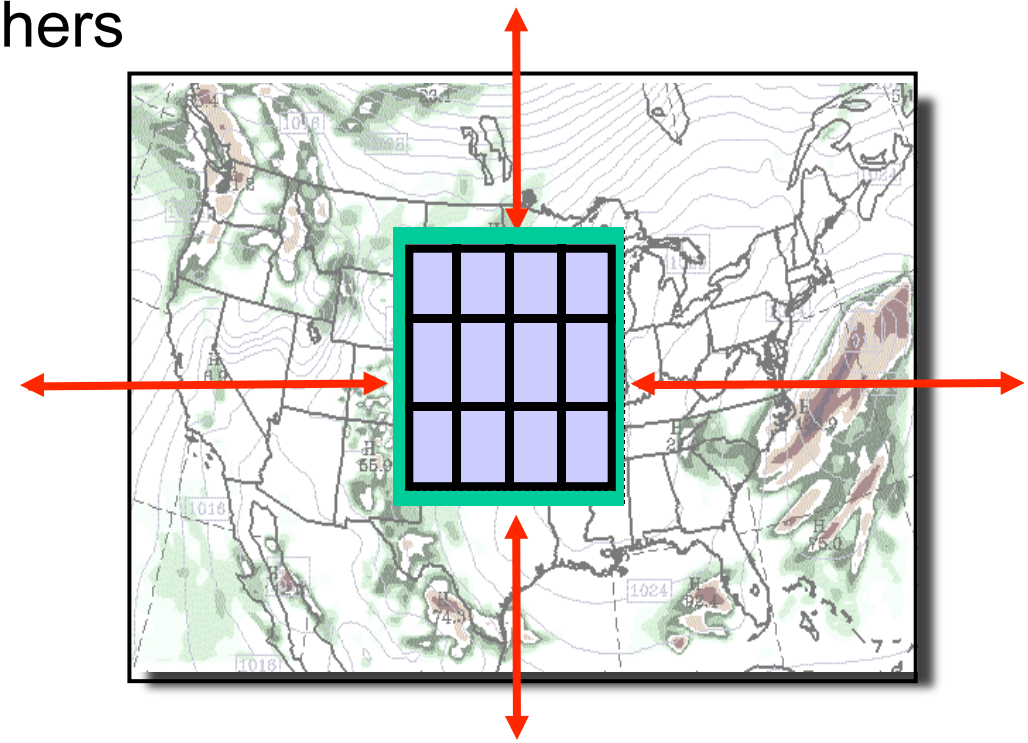
- Halo updates





Distributed Memory (MPI) Communications

- Halo updates
- Periodic boundary updates
- Parallel transposes
- Nesting scatters/gathers

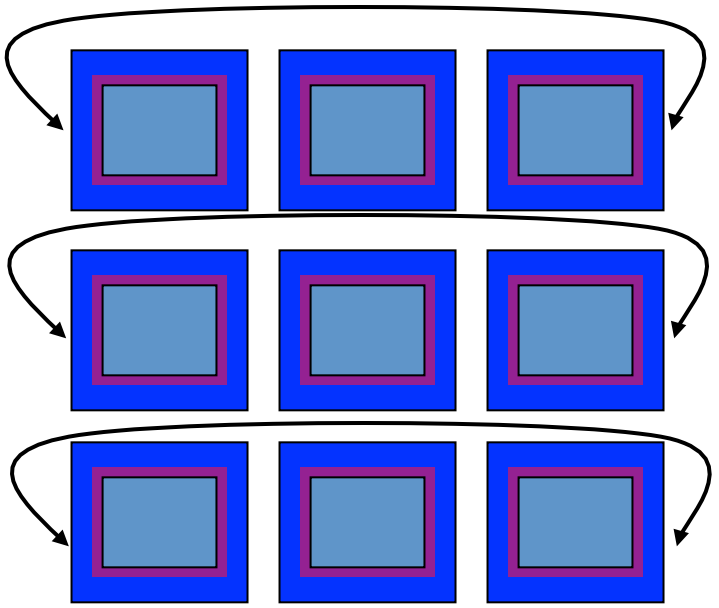


APPLICATION

SYSTEM

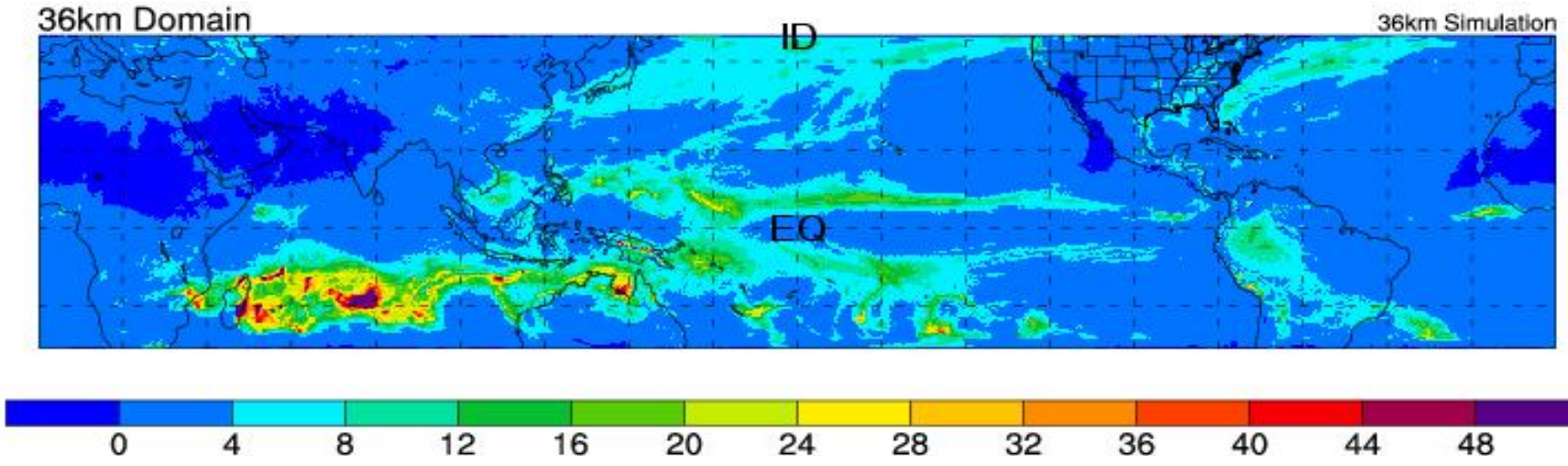
HARDWARE

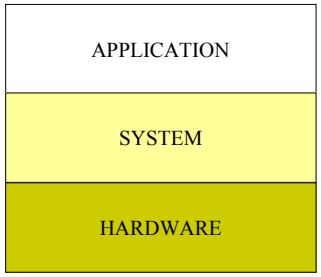
Distributed Memory (MPI) Communications



- Halo updates
- Periodic boundary updates
- Parallel transposes
- Nesting scatters/gathers

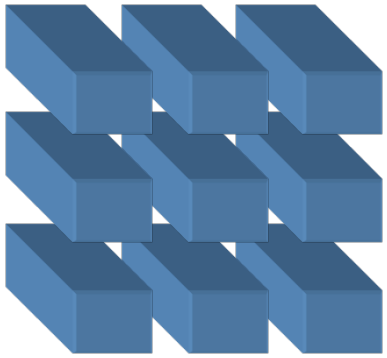
Average Daily Total rainfall (mm) - March 1997



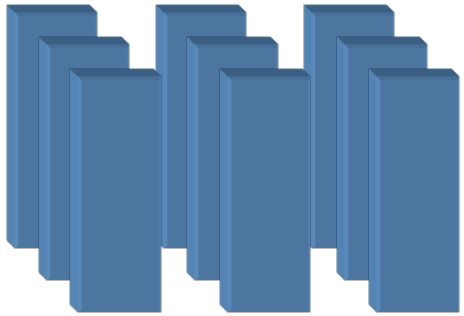


Distributed Memory (MPI) Communications

- Halo updates
- Periodic boundary updates
- Parallel transposes
- Nesting scatters/gathers

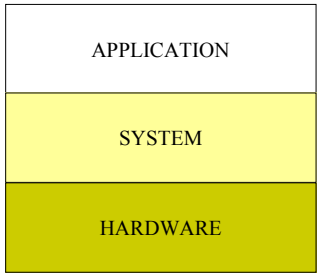


all y on patch



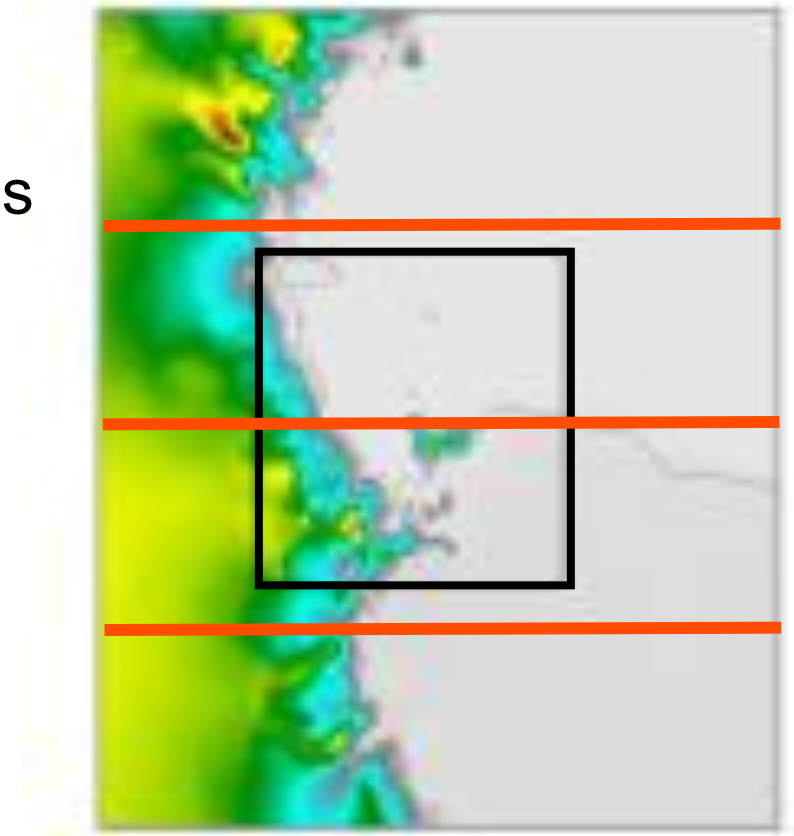
all z on patch

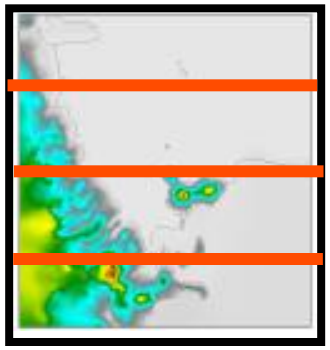
all x on patch



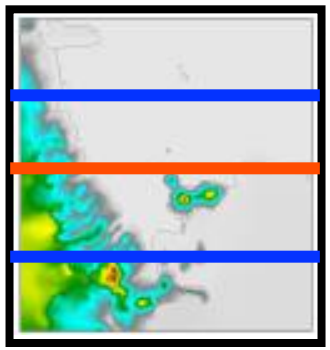
Distributed Memory (MPI) Communications

- Halo updates
- Periodic boundary updates
- Parallel transposes
- Nesting scatters/gathers





NEST:2.22 km



INTERMEDIATE: 6.66 km

COARSE
Ross Island
6.66 km

WRF Model Layer Interface

```
SUBROUTINE driver_for_some_physics_suite (
    . . .
!$OMP DO PARALLEL
   DO ij = 1, numtiles
      its = i_start(ij) ; ite = i_end(ij)
      jts = j_start(ij) ; jte = j_end(ij)
      CALL model_subroutine( arg1, arg2, . . .
           ids , ide , jds , jde , kds , kde ,
           ims , ime , jms , jme , kms , kme ,
           its , ite , jts , jte , kts , kte )
   END DO
    . . .

 END SUBROUTINE
```

WRF Model Layer Interface

template for model layer subroutine

```
SUBROUTINE model_subroutine ( &
  arg1, arg2, arg3, … , argn,   &
  ids, ide, jds, jde, kds, kde, &   ! Domain dims
  ims, ime, jms, jme, kms, kme, &   ! Memory dims
  its, ite, jts, jte, kts, kte  )   ! Tile dims

IMPLICIT NONE

! Define Arguments (State and I1) data
REAL, DIMENSION (ims:ime,kms:kme,jms:jme) :: arg1, . . .
REAL, DIMENSION (ims:ime,jms:jme)         :: arg7, . . .
 . . .
! Define Local Data (I2)
REAL, DIMENSION (its:ite,kts:kte,jts:jte) :: loc1, . . .
 . . .
```

WRF Model Layer Interface

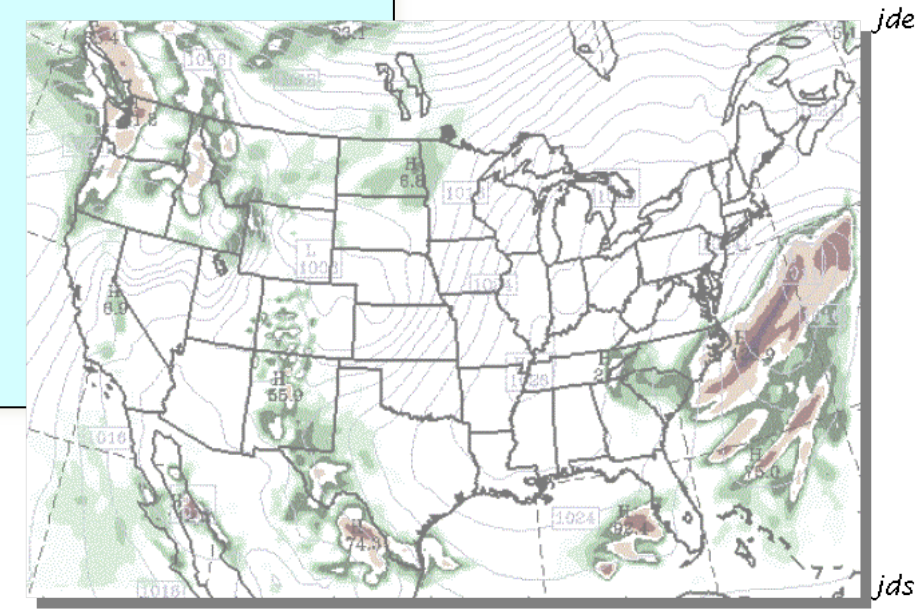
template for model layer subroutine

```
. . .
 ! Executable code; loops run over tile
 ! dimensions
 DO j = jts, MIN(jte,jde-1)
   DO k = kts, kte
     DO i = its, MIN(ite,ide-1)
       loc1(i,k,j) = arg1(i,k,j) + …
     END DO
   END DO
 END DO
```

template for model layer subroutine

```
SUBROUTINE model ( &
  arg1, arg2, arg3, … , argn,    &
  ids, ide, jds, jde, kds, kde, &   ! Domain dims
  ims, ime, jms, jme, kms, kme, &   ! Memory dims
  its, ite, jts, jte, kts, kte  )   ! Tile dims

IMPLICIT NONE

! Define Arguments (S and I1) data
REAL, DIMENSION (ims:ime,kms:kme,jms:jme) :: arg1, . . .
REAL, DIMENSION (ims:ime,jms:jme)         :: arg7, . . .
 . . .
! Define Local Data (I2)
REAL, DIMENSION (its:ite,kts:kte,jts:jte) :: loc1, . . .
 . . .
! Executable code; loops run over tile
! dimensions
DO j = MAX(jts,jds), MIN(jte,jde-1)
  DO k = kts, kte
    DO i = MAX(its,ids), MIN(ite,ide-1)
      loc1(i,k,j) = arg1(i,k,j) + …
    END DO
  END DO
END DO
```

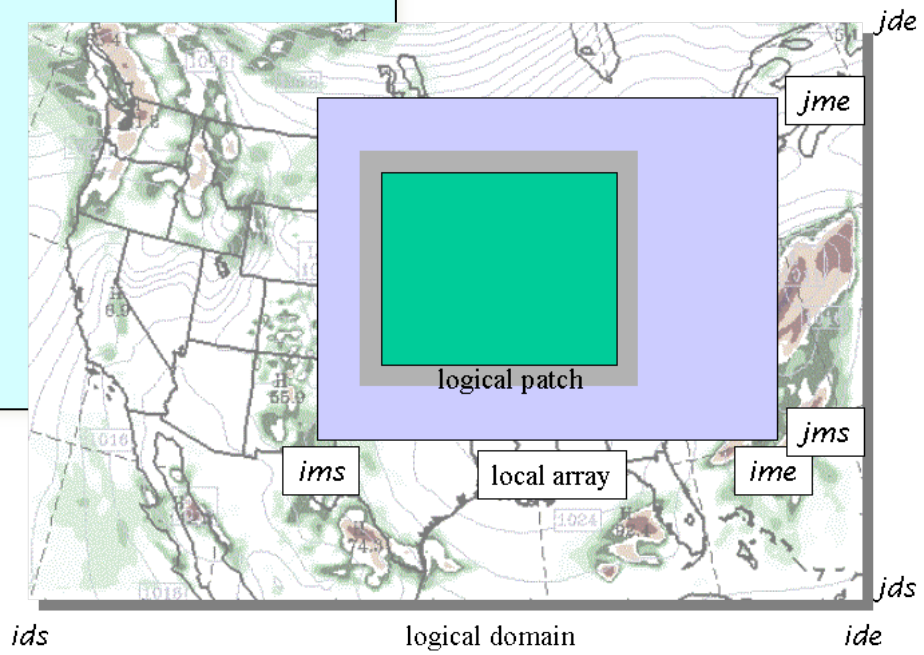
- Domain dimensions
  - Size of logical domain
  - Used for bdy tests, etc.



template for model layer subroutine

```
SUBROUTINE model ( &
  arg1, arg2, arg3, … , argn,    &
  ids, ide, jds, jde, kds, kde, &   ! Domain dims
  ims, ime, jms, jme, kms, kme, &   ! Memory dims
  its, ite, jts, jte, kts, kte  )   ! Tile dims

IMPLICIT NONE

! Define Arguments (S and I1) data
REAL, DIMENSION (ims:ime,kms:kme,jms:jme) :: arg1, . . .
REAL, DIMENSION (ims:ime,jms:jme)         :: arg7, . . .
 . . .
! Define Local Data (I2)
REAL, DIMENSION (its:ite,kts:kte,jts:jte) :: loc1, . . .
 . . .
! Executable code; loops run over tile
! dimensions
DO j = MAX(jts,jds), MIN(jte,jde-1)
  DO k = kts, kte
    DO i = MAX(its,ids), MIN(ite,ide-1)
      loc1(i,k,j) = arg1(i,k,j) + …
    END DO
  END DO
END DO
```

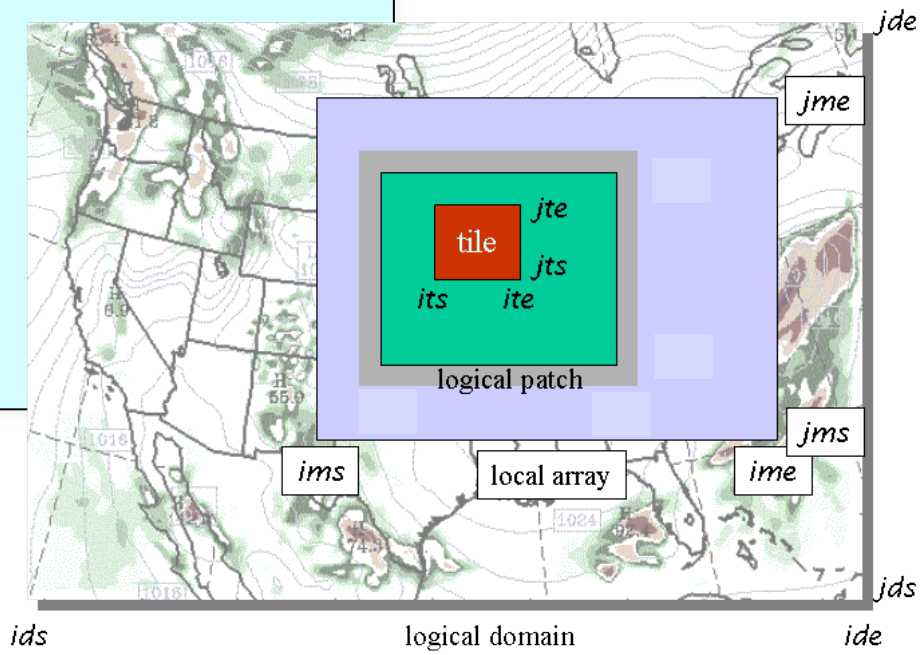
- Domain dimensions
  - Size of logical domain
  - Used for bdy tests, etc.
- Memory dimensions
  - Used to dimension dummy arguments
  - Do not use for local arrays



template for model layer subroutine

```
SUBROUTINE model ( &
  arg1, arg2, arg3, … , argn,    &
  ids, ide, jds, jde, kds, kde, &   ! Domain dims
  ims, ime, jms, jme, kms, kme, &   ! Memory dims
  its, ite, jts, jte, kts, kte  )   ! Tile dims

IMPLICIT NONE

! Define Arguments (S and I1) data
REAL, DIMENSION (ims:ime,kms:kme,jms:jme) :: arg1, . . .
REAL, DIMENSION (ims:ime,jms:jme)         :: arg7, . . .
 . . .
! Define Local Data (I2)
REAL, DIMENSION (its:ite,kts:kte,jts:jte) :: loc1, . . .
 . . .
! Executable code; loops run over tile
! dimensions
DO j = MAX(jts,jds), MIN(jte,jde-1)
  DO k = kts, kte
    DO i = MAX(its,ids), MIN(ite,ide-1)
      loc1(i,k,j) = arg1(i,k,j) + …
    END DO
  END DO
END DO
```

- Domain dimensions
  - Size of logical domain
  - Used for bdy tests, etc.
- Memory dimensions
  - Used to dimension dummy arguments
  - Do not use for local arrays
- Tile dimensions
  - Local loop ranges
  - Local array dimensions

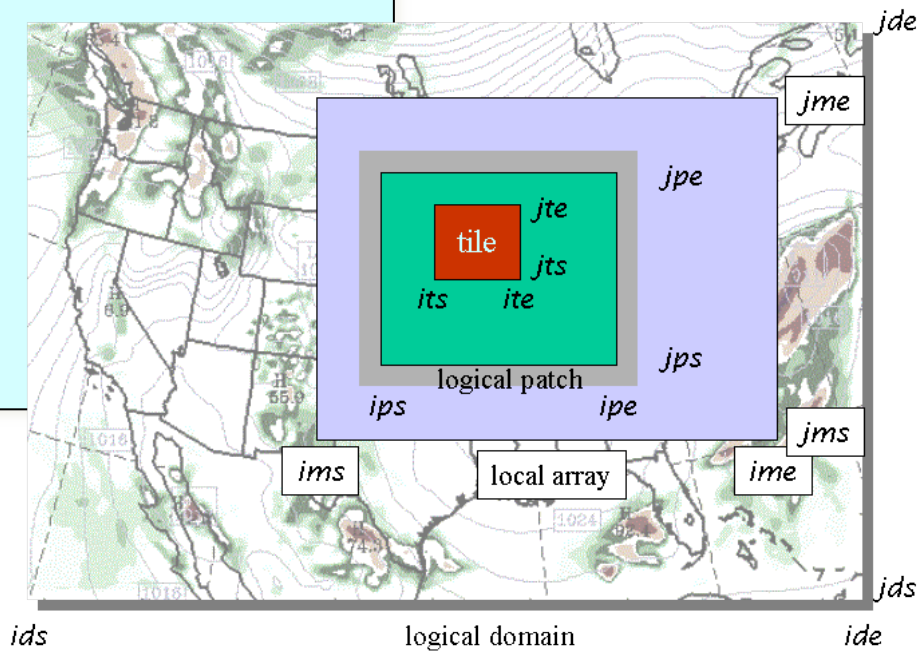


template for model layer subroutine

```
SUBROUTINE model ( &
  arg1, arg2, arg3, … , argn,    &
  ids, ide, jds, jde, kds, kde, &   ! Domain dims
  ims, ime, jms, jme, kms, kme, &   ! Memory dims
  its, ite, jts, jte, kts, kte  )   ! Tile dims

IMPLICIT NONE

! Define Arguments (S and I1) data
REAL, DIMENSION (ims:ime,kms:kme,jms:jme) :: arg1, . . .
REAL, DIMENSION (ims:ime,jms:jme)         :: arg7, . . .
 . . .
! Define Local Data (I2)
REAL, DIMENSION (its:ite,kts:kte,jts:jte) :: loc1, . . .
 . . .
! Executable code; loops run over tile
! dimensions
DO j = MAX(jt,jds), MIN(jte,jde-1)
  DO k = kts, kte
    DO i = MAX(its,ids), MIN(ite,ide-1)
      loc1(i,k,j) = arg1(i,k,j) + …
    END DO
  END DO
END DO
```

- Domain dimensions
  - Size of logical domain
  - Used for bdy tests, etc.
- Memory dimensions
  - Used to dimension dummy arguments
  - Do not use for local arrays
- Tile dimensions
  - Local loop ranges
  - Local array dimensions

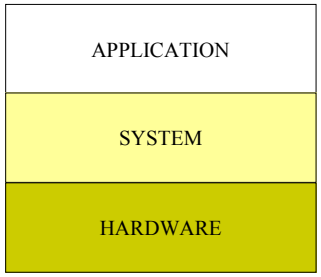
- Patch dimensions
  - Start and end indices of local distributed memory subdomain
  - Available from mediation layer (solve) and driver layer; not usually needed or used at model layer



How OpenMP and MPI Fit

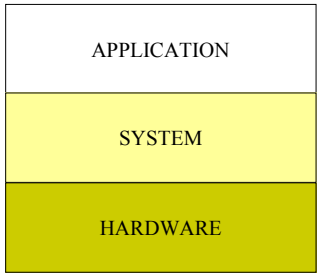
WRF Domain Decomposition

- As you increase the number of total MPI tasks, you reduce the amount of work inside of each MPI task
- The amount of time to process communication between MPI tasks tends to be ***at best*** constant
- As more MPI tasks are involved, more contention for hardware resources due to communication is likely
- As the computation time gets smaller compared to the communications time, parallel efficiency suffers

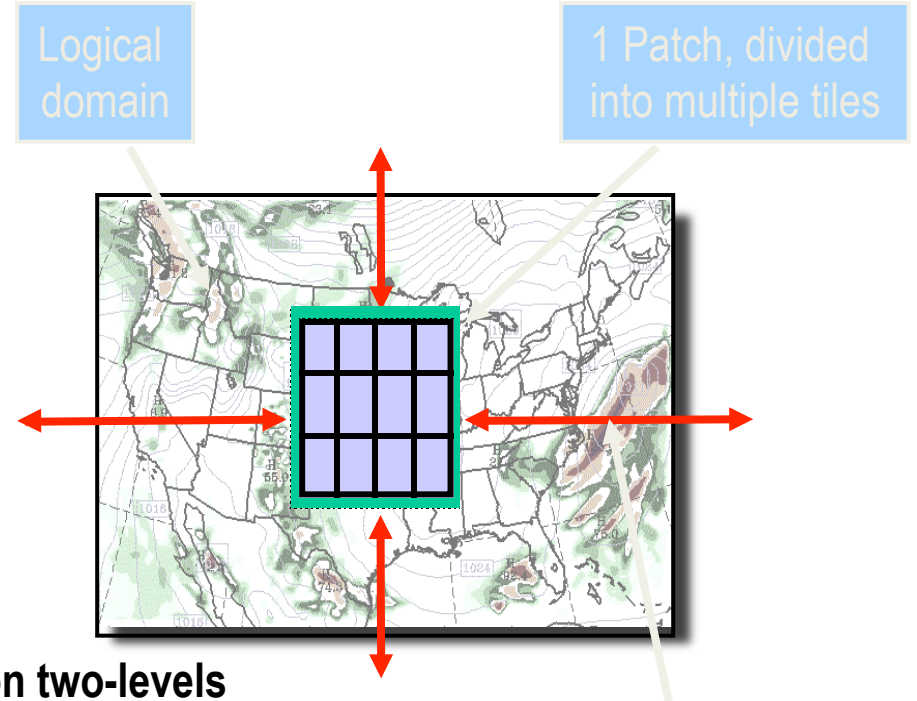


Application: WRF

- WRF can be run serially or as a parallel job
- WRF uses *domain decomposition* to divide total amount of work over parallel processes



Parallelism in WRF: Multi-level Decomposition



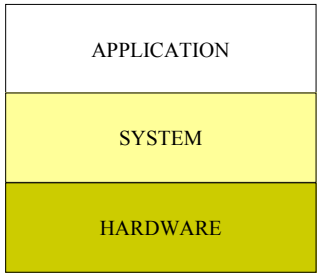
- **Single version of code for efficient execution on:**
  - Distributed-memory
  - Shared-memory (SMP)
  - Clusters of SMPs
  - Vector and microprocessors

Model domains are decomposed for parallelism on two-levels

Patch: section of model domain allocated to a distributed memory node, this is the scope of a mediation layer solver or physics driver.

Tile: section of a patch allocated to a shared-memory processor within a node; this is also the scope of a model layer subroutine.

Distributed memory parallelism is over patches; shared memory parallelism is over tiles within patches



Hardware: The Computer

- The ‘N’ in NWP
- Components
  - Processor
    - A program counter
    - Arithmetic unit(s)
    - Some scratch space (registers)
    - Circuitry to store/retrieve from memory device
    - Cache
  - Memory
  - Secondary storage
  - Peripherals
- The implementation has been continually refined, but the basic idea hasn’t changed much

APPLICATION

SYSTEM

HARDWARE

Hardware has not changed much…

A computer in 1960

IBM 7090

6-way superscalar

36-bit floating point precision

~144 Kbytes

~50,000 flop/s
48hr 12km WRF CONUS in 600 years

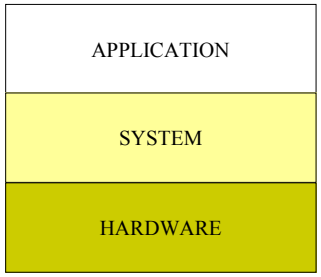
A computer in 2013

Dual core, 2.6 GHz chip

64-bit floating point precision

20 MB L3

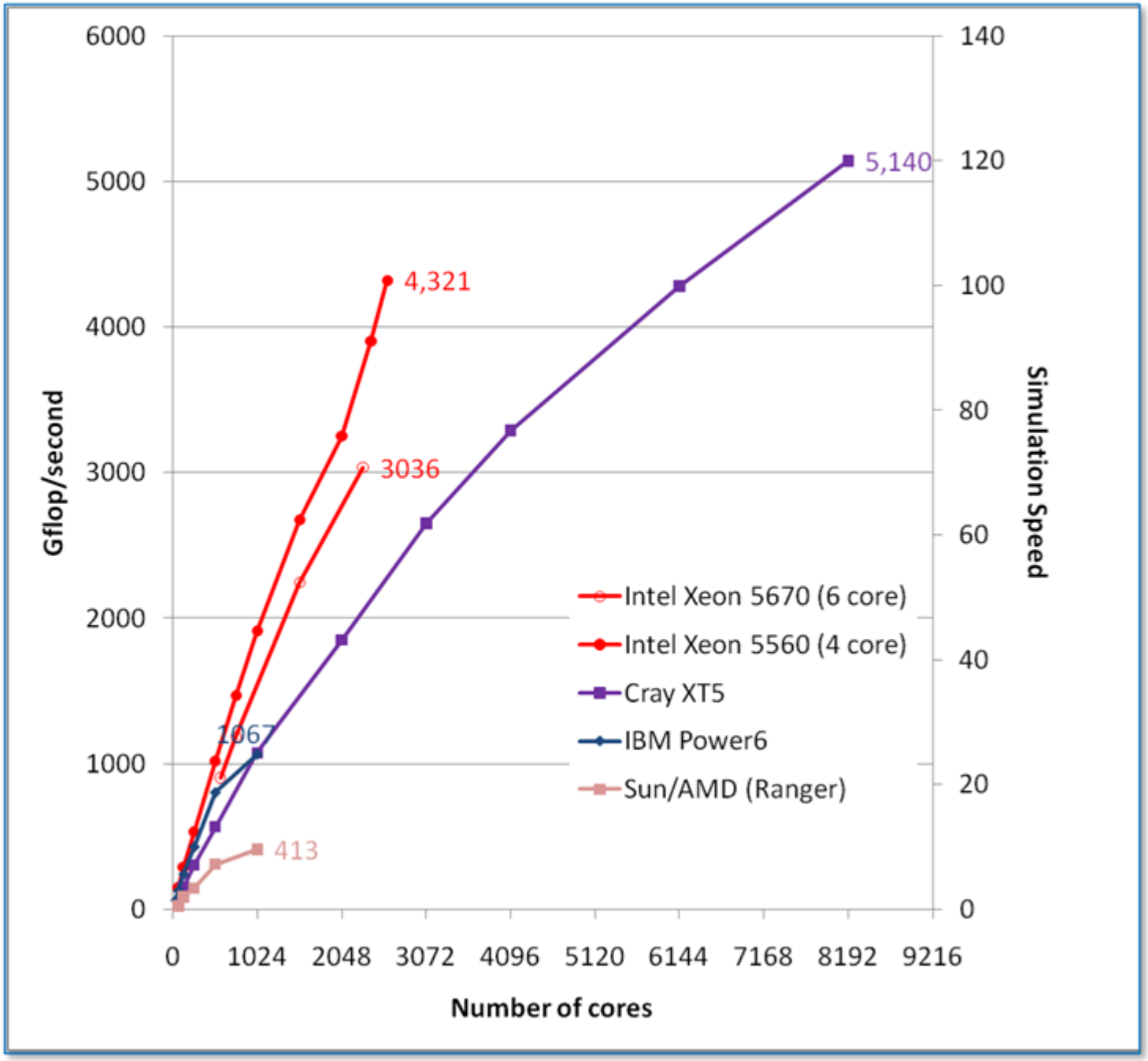
~5,000,000,000 flop/s
48 12km WRF CONUS in 26 Hours



…how we use it has

- Fundamentally, processors haven't changed much since 1960
- Quantitatively, they haven't improved nearly enough
  - 100,000x increase in peak speed
  - 100,000x increase in memory size
- We make up the difference with parallelism
  - Ganging multiple processors together to achieve $10^{11\text{-}12}$ flop/second
  - Aggregate available memories of $10^{11\text{-}12}$ bytes

~1,000,000,000,000 flop/s ~2500 procs
48-h,12-km WRF CONUS in under 15 minutes


6000
5000
4000
3000
2000
1000
0
Gflop/second
140
120
100
80
60
40
20
0
Simulation Speed
0
1024
2048
3072
4096
5120
6144
7168
8192
9216
Number of cores
5,140
4,321
3036
1067
413
Intel Xeon 5670 (6 core)
Intel Xeon 5560 (4 core)
Cray XT5
IBM Power6
Sun/AMD (Ranger)

Strong vs Weak Scaling

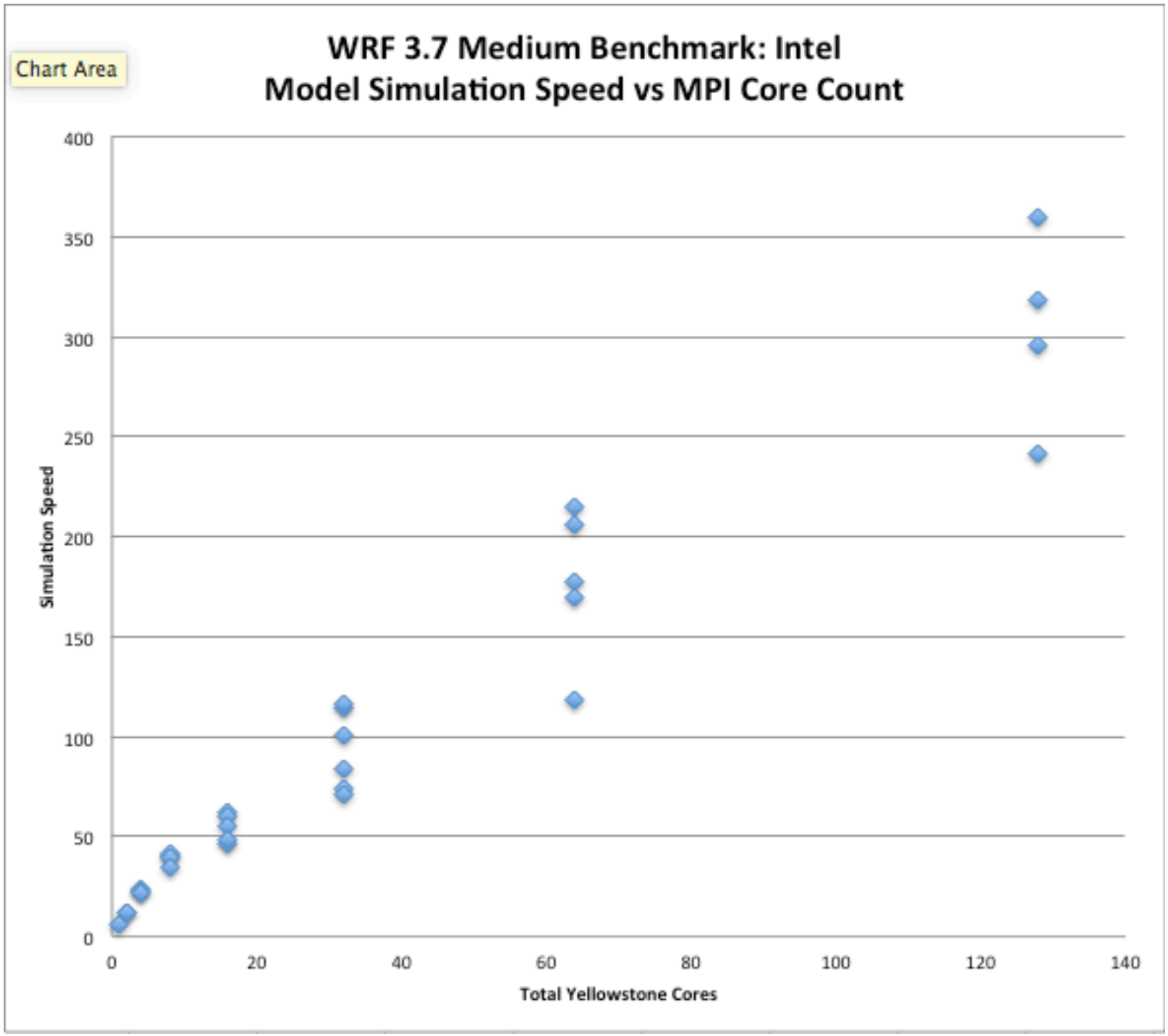
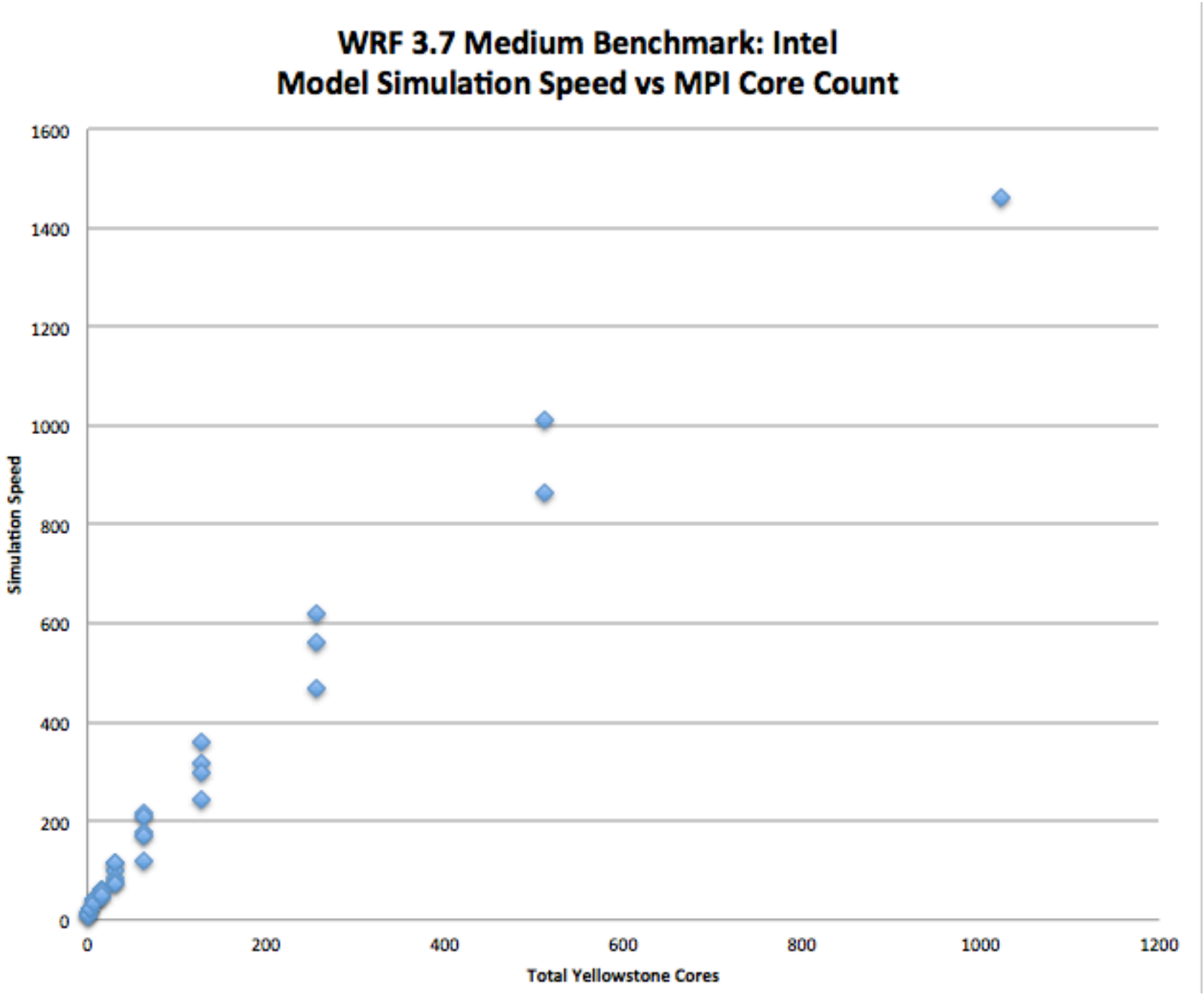


Chart Area
WRF 3.7 Medium Benchmark: Intel
Model Simulation Speed vs MPI Core Count
Simulation Speed
0
50
100
150
200
250
300
350
400
0
20
40
60
80
100
120
140
Total Yellowstone Cores


WRF 3.7 Medium Benchmark: Intel
Model Simulation Speed vs MPI Core Count
Simulation Speed
1600
1400
1200
1000
800
600
400
200
0
0
200
400
600
800
1000
1200
Total Yellowstone Cores

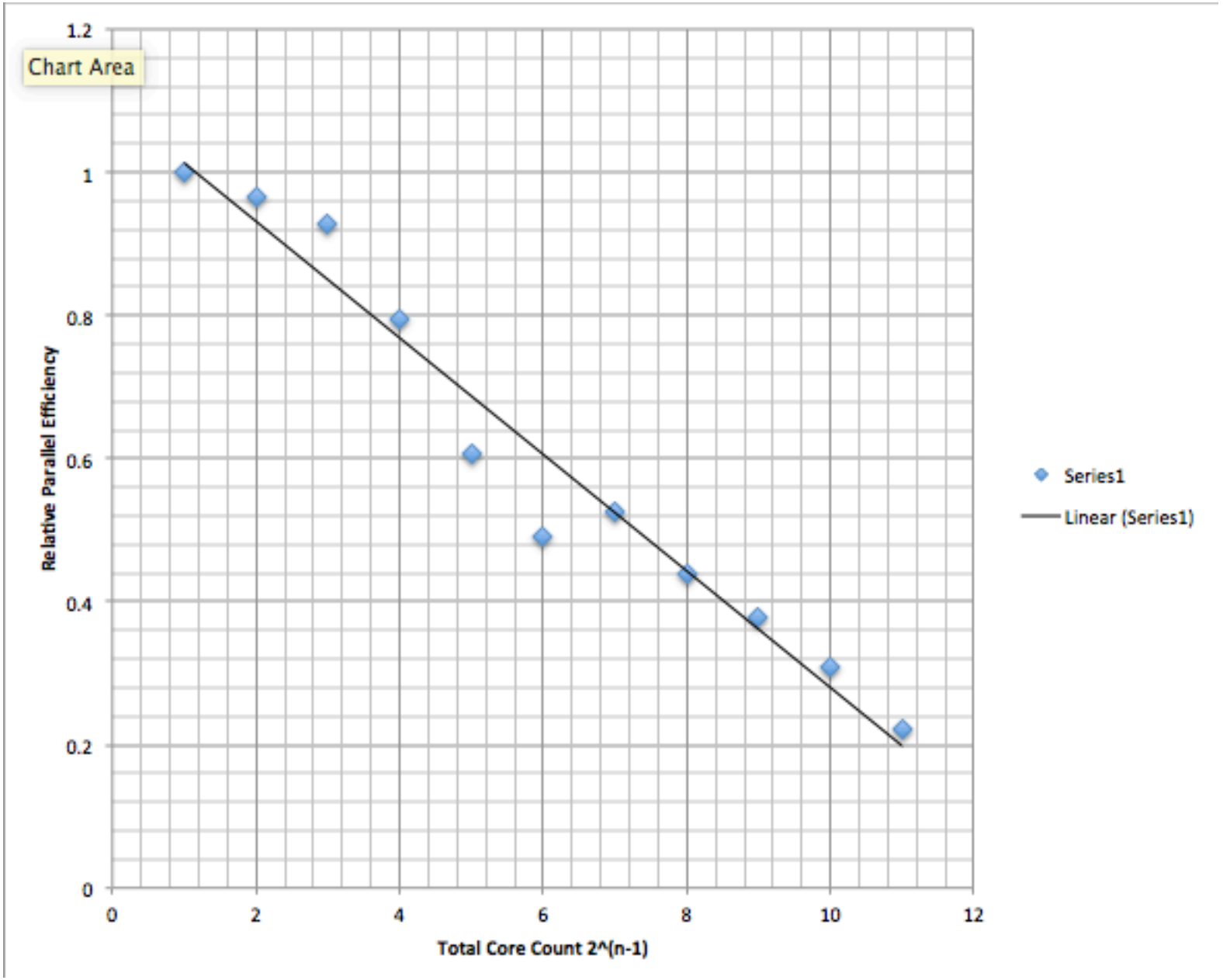
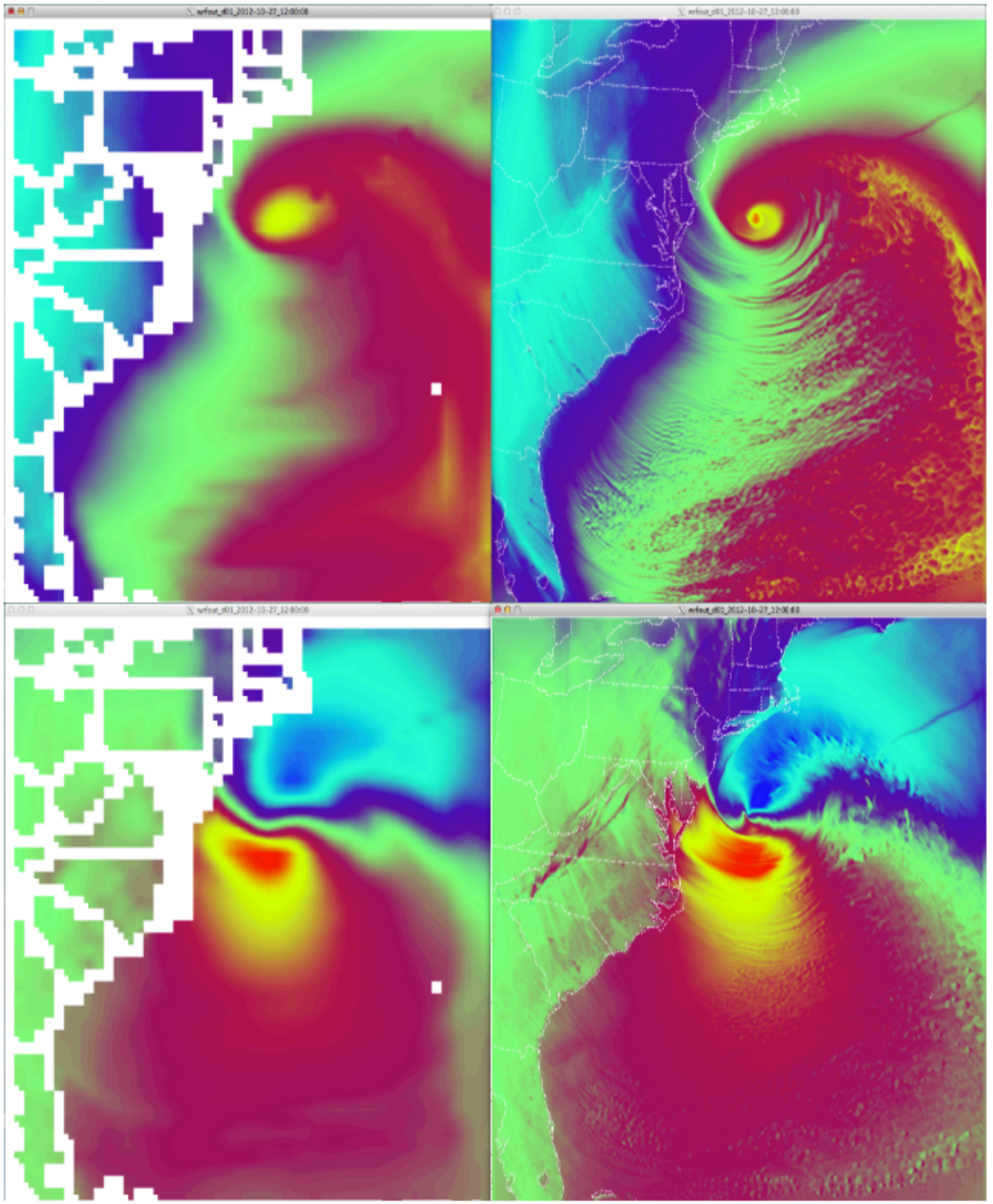


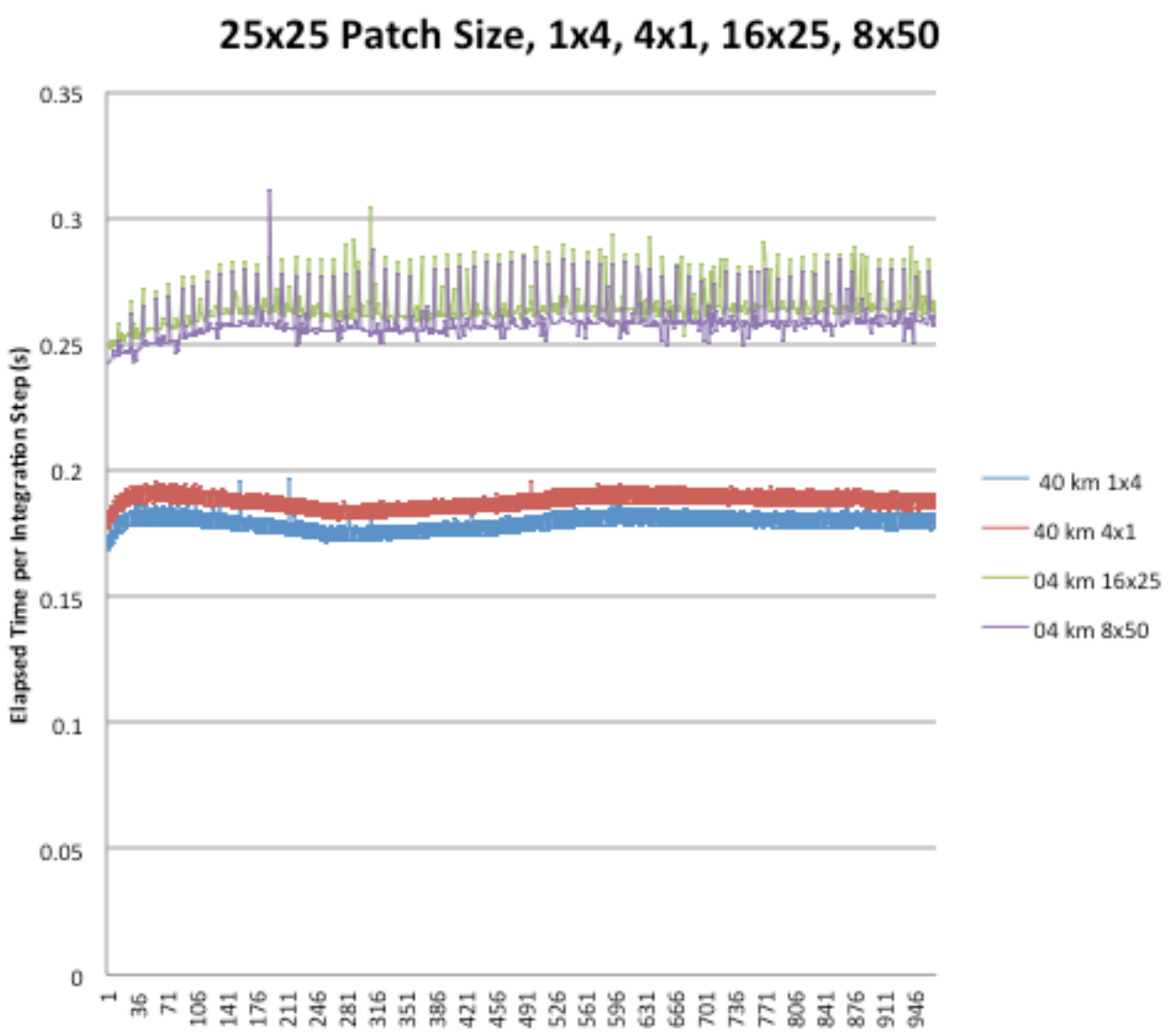
Chart Area
1.2
1
0.8
0.6
0.4
0.2
0
Relative Parallel Efficiency
0
2
4
6
8
10
12
Total Core Count 2^(n-1)
Series1
Linear (Series1)

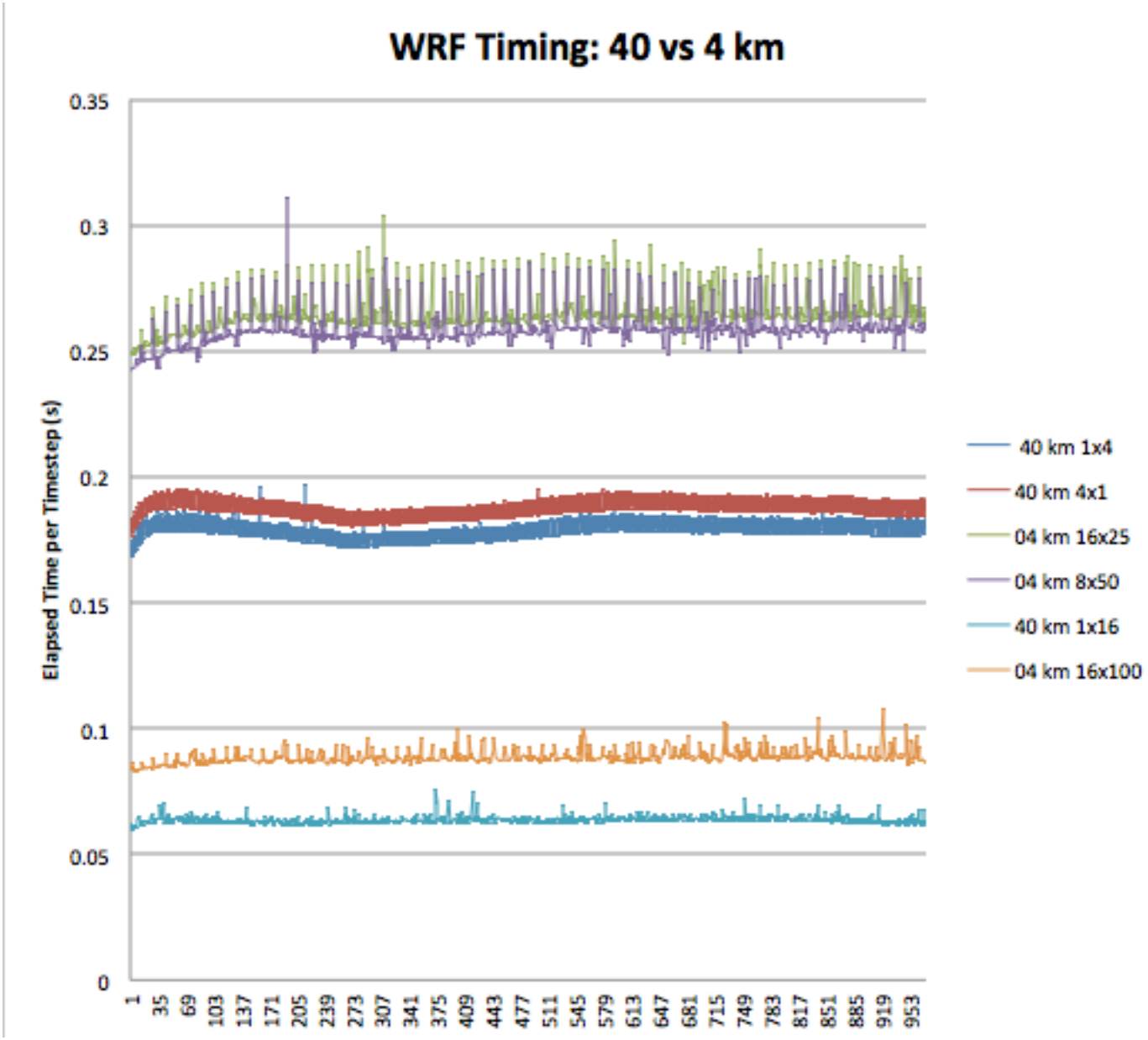
- Hurricane Sandy example

- 40 km 50x50
- Decomposed 4 cores (1x4, 4x1)

- 4 km 500x500
- Decomposed 400 cores (16x25, 8x50)

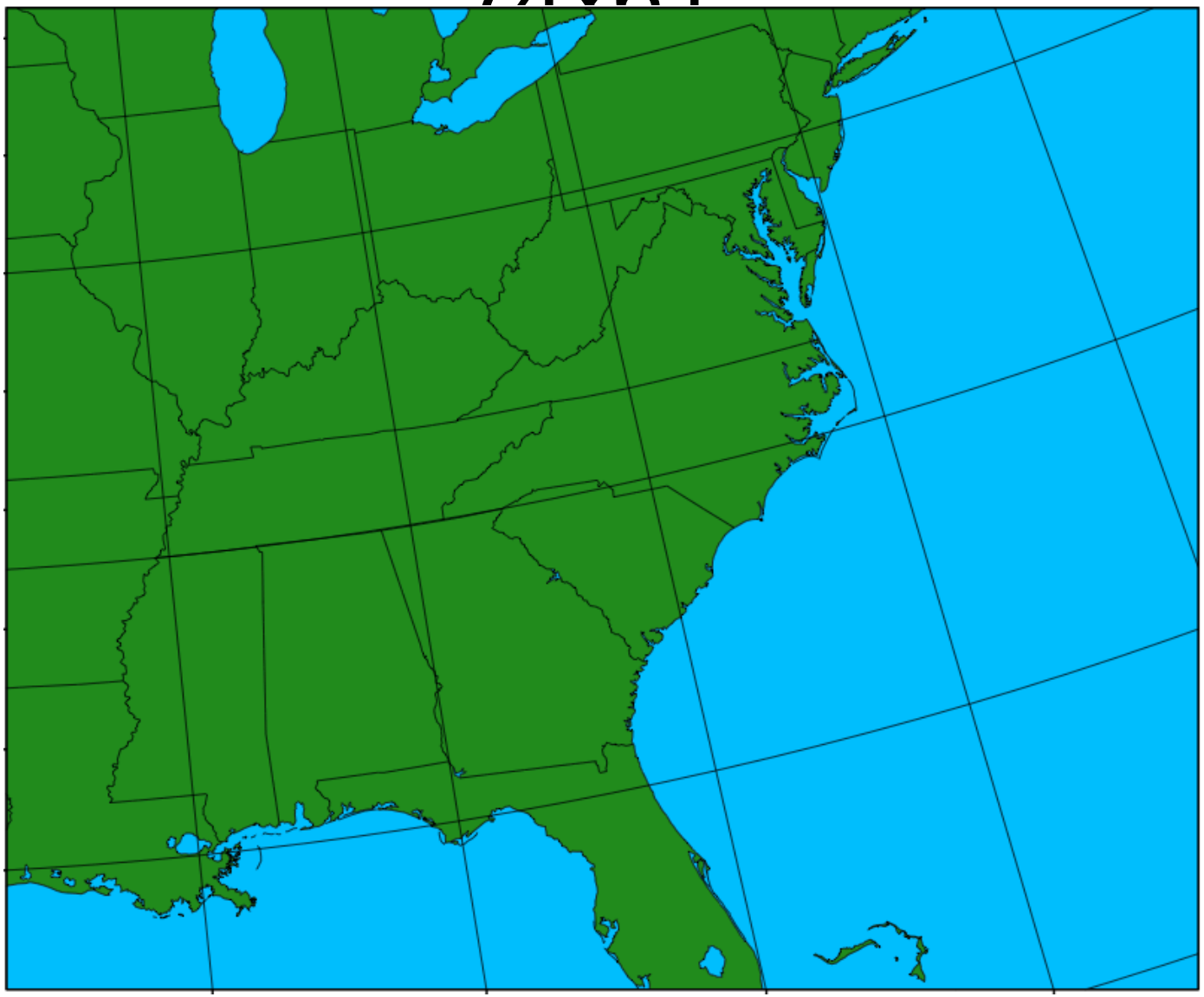



25x25 Patch Size, 1x4, 4x1, 16x25, 8x50
Elapsed Time per Integration Step (s)
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1
36
71
106
141
176
211
246
281
316
351
386
421
456
491
526
561
596
631
666
701
736
771
806
841
876
911
946
40 km 1x4
40 km 4x1
04 km 16x25
04 km 8x50

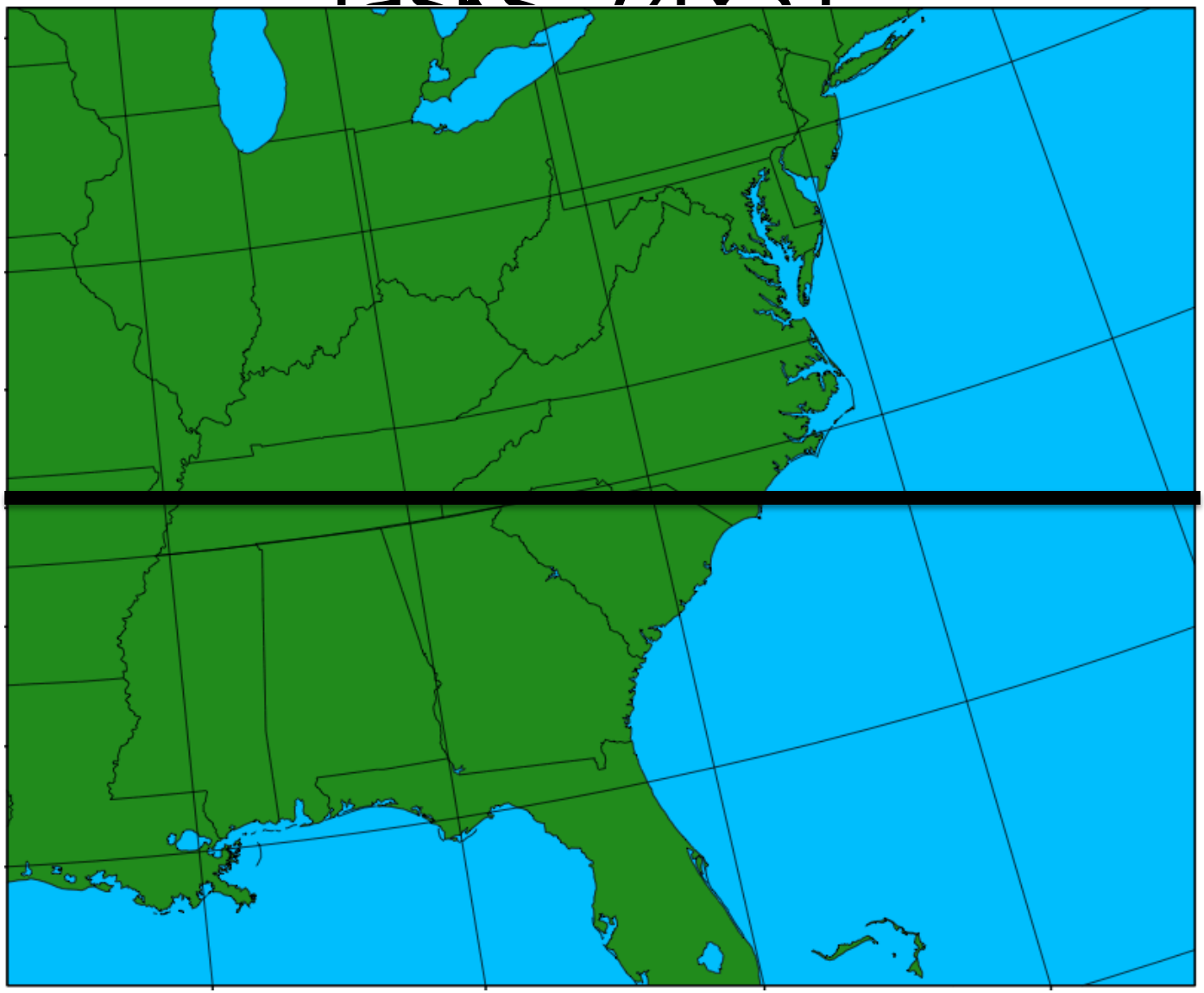

WRF Timing: 40 vs 4 km
Elapsed Time per Timestep (s)
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1
35
69
103
137
171
205
239
273
307
341
375
409
443
477
511
545
579
613
647
681
715
749
783
817
851
885
919
953
40 km 1x4
40 km 4x1
04 km 16x25
04 km 8x50
40 km 1x16
04 km 16x100

Choosing Domains

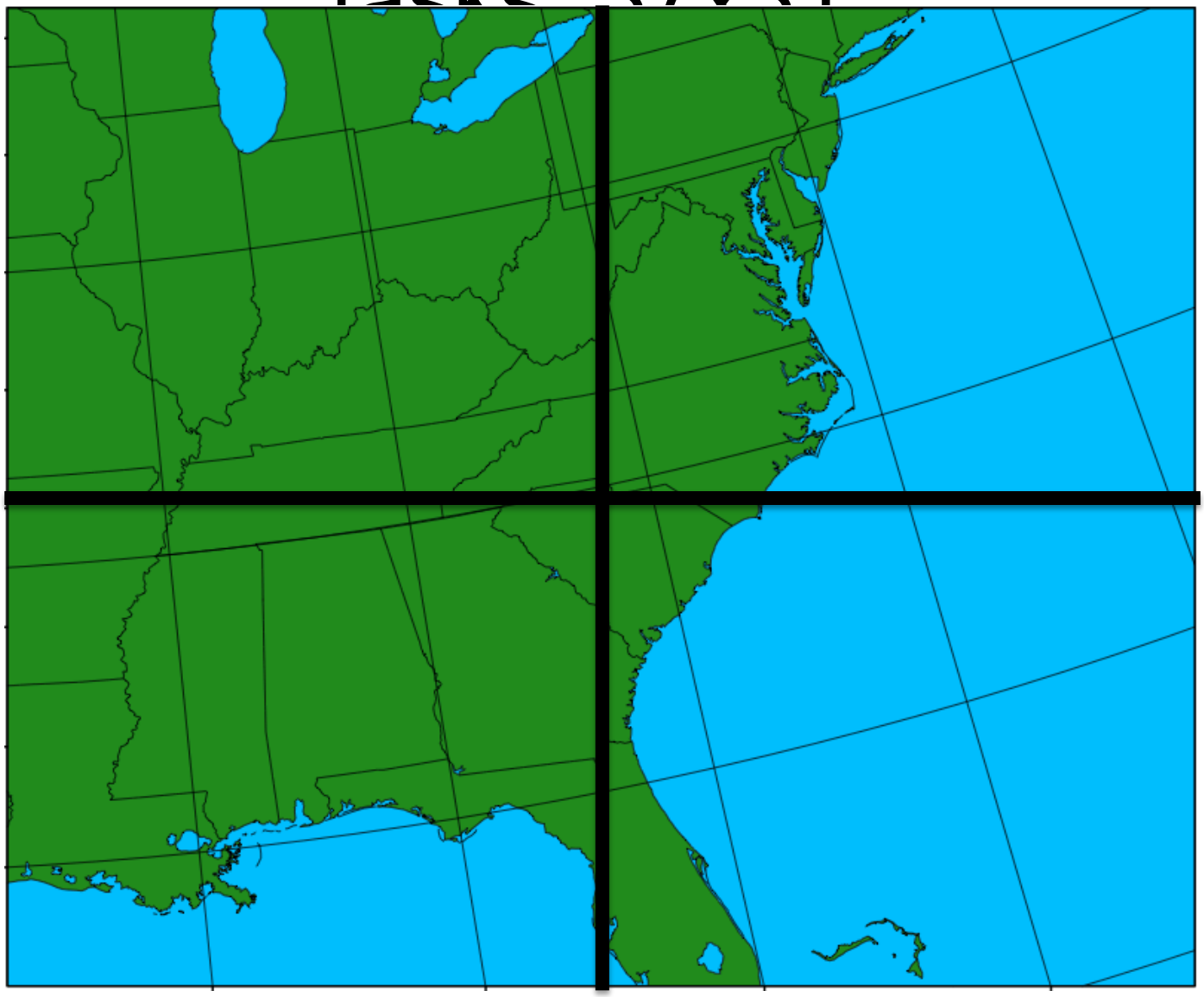
January 2000 Benchmark – 1 task: 74x61



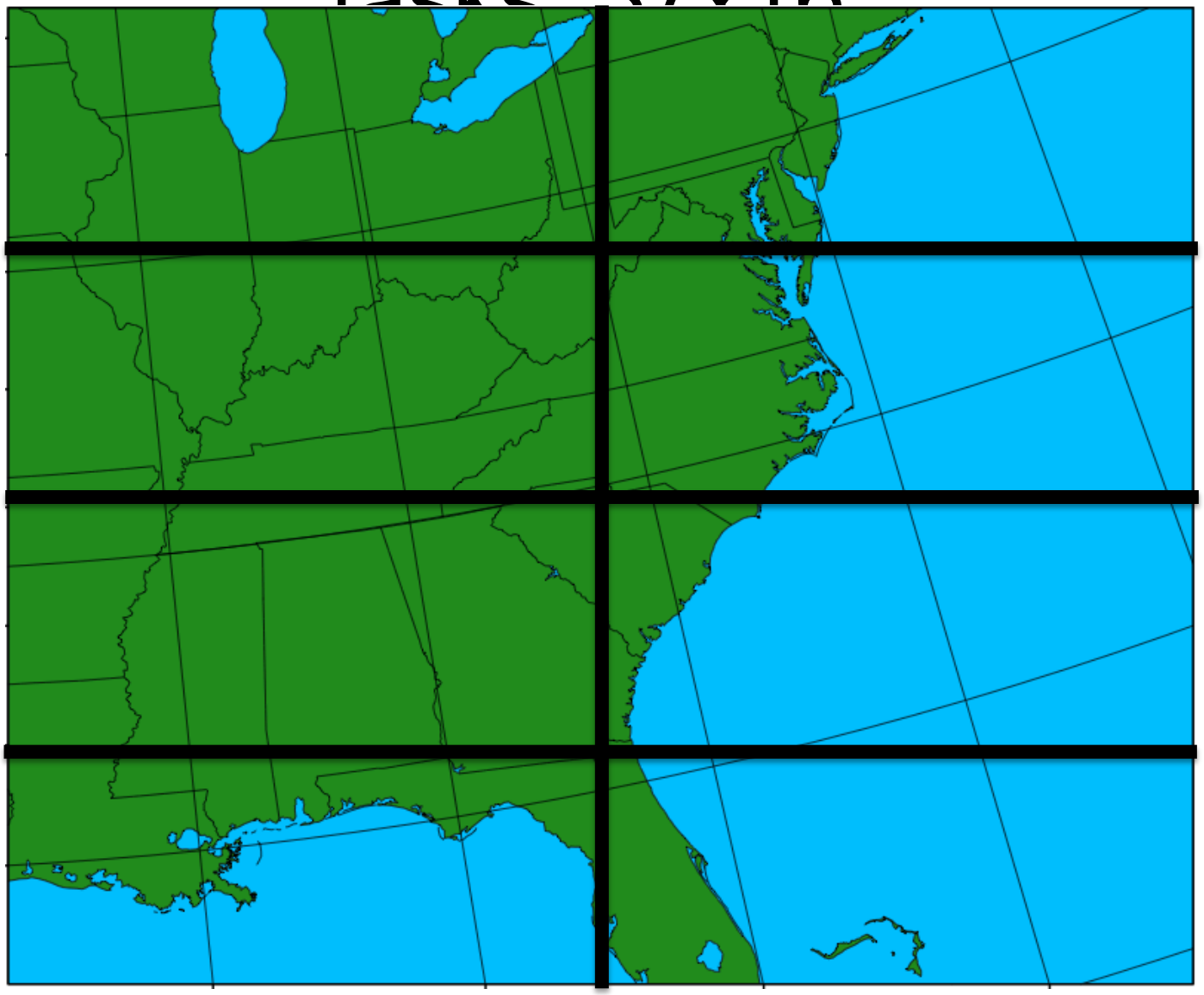
January 2000 Benchmark – 2 tasks: 74x31



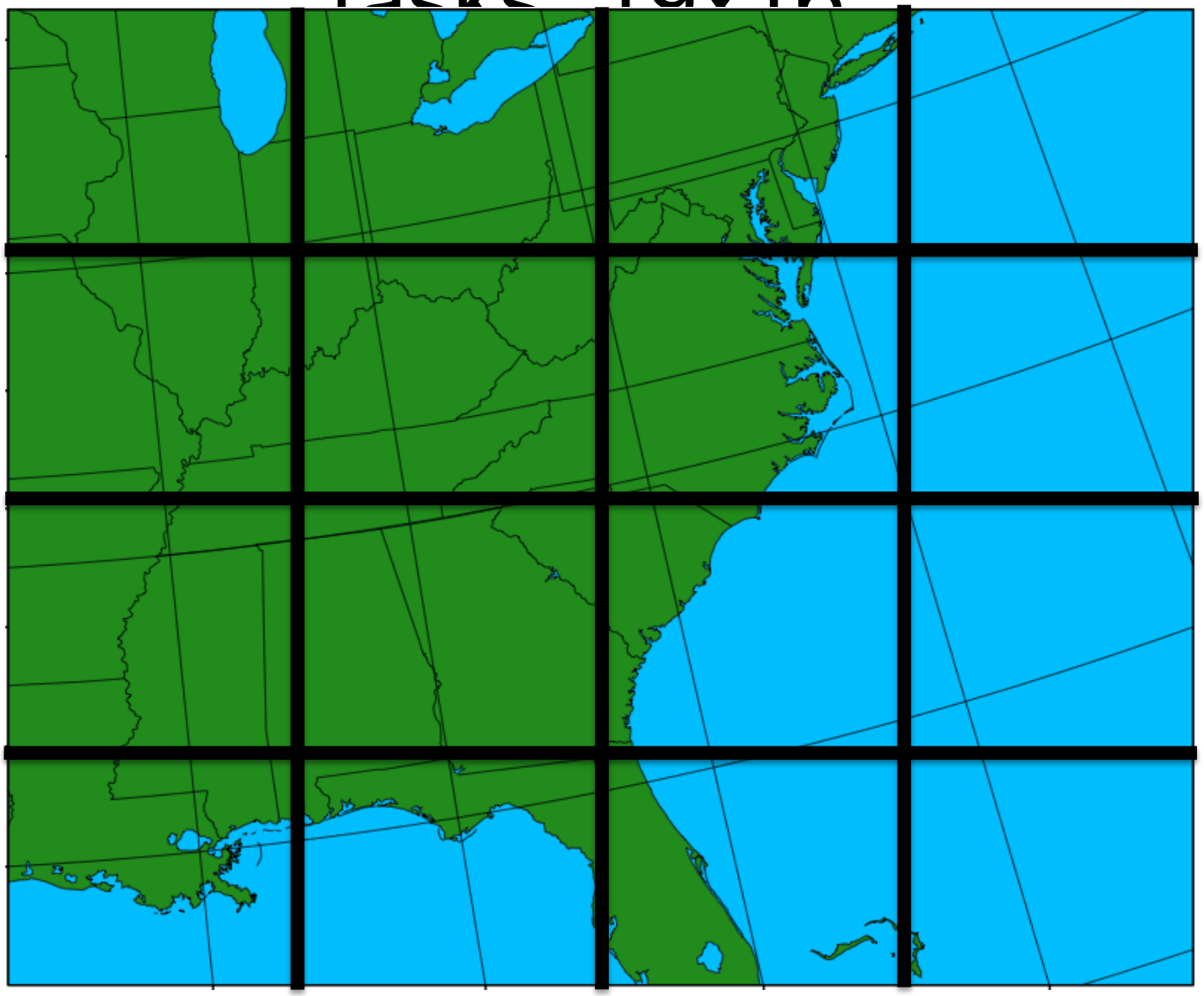
January 2000 Benchmark – 4 tasks: 37x31



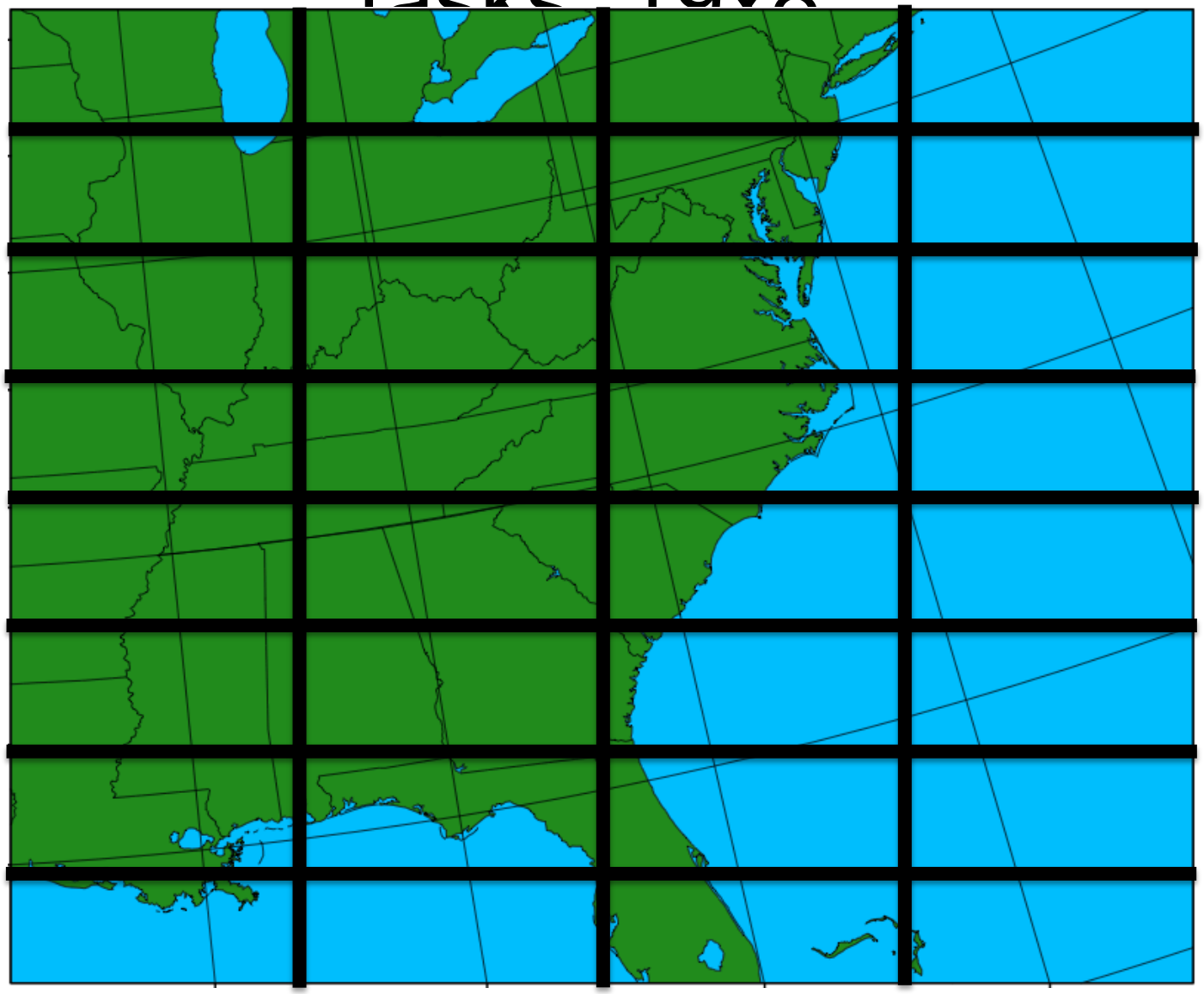
January 2000 Benchmark – 8 tasks: 37x16



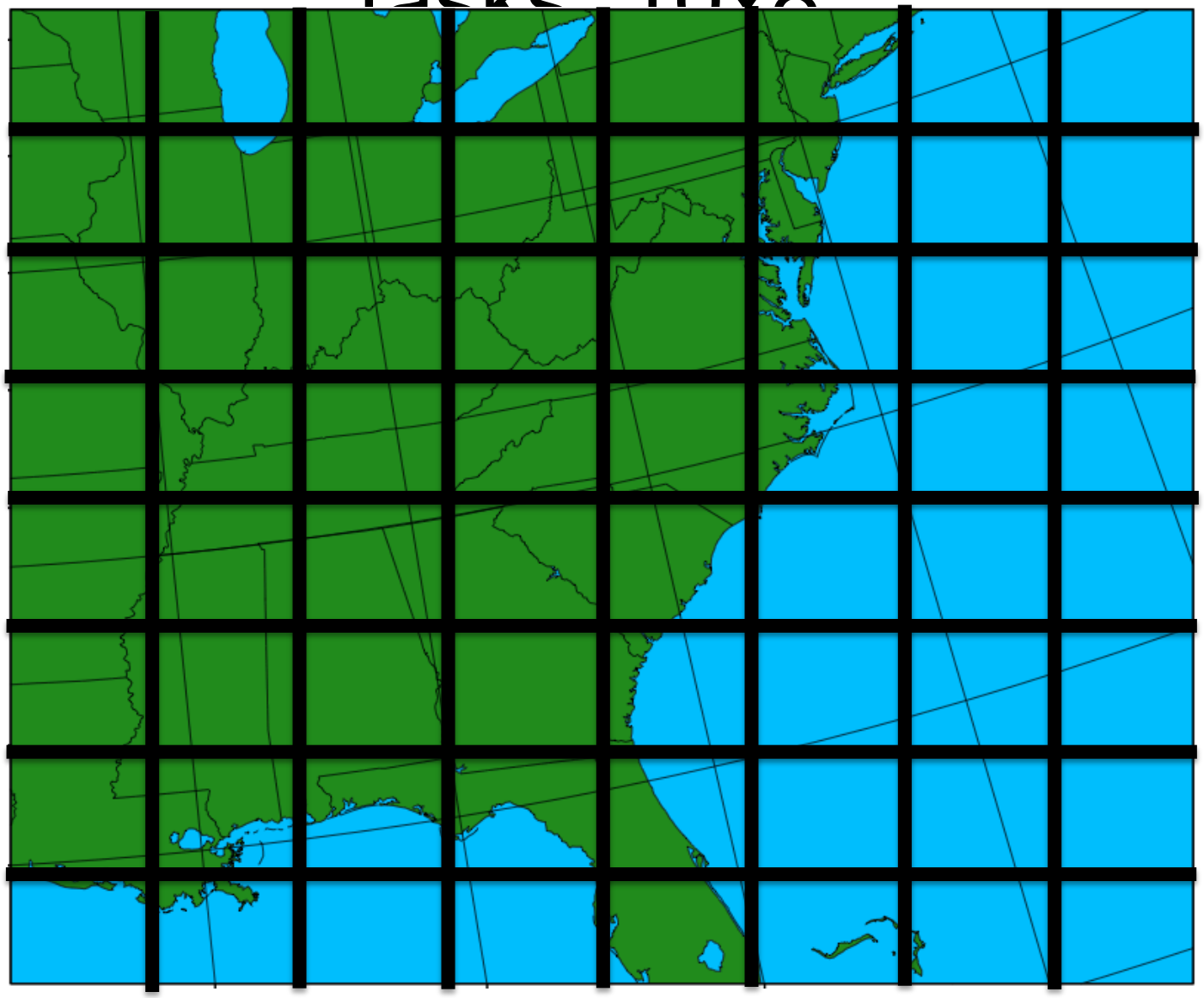
January 2000 Benchmark – 16 tasks: 19x16



January 2000 Benchmark – 32 tasks: 19x8



January 2000 Benchmark – 64 tasks: 10x8



Choosing Computational Domains

- What does it mean to choose an appropriate computational domain?

- For the WRF model to run correctly and yield physically meaningful results, a geographic domain must be selected that allows the model to the necessary information to develop the simulation.

Choosing Computational Domains

- What does it mean to choose an appropriate computational domain?

- Similarly, the computationally invariant aspects of the domain layout are important. For the same physics options and domain size, a simulation over the US vs Europe take about the same amount of time.

Choosing Computational Domains

- What does it mean to choose an appropriate computational domain?

- The computational domain is concerned with numbers of grid cells and the mapping of those cells onto processors.

Choosing Computational Domains

- What does it mean to choose an appropriate computational domain?

- The WRF model by default has a simple algorithm to divvy up the domain into patch sized pieces.

WRF Domain Decomposition

- The WRF model decomposes domains horizontally
- For *n* MPI tasks, the two nearest factors (*n= k * m*)are selected; the larger is used to decompose the y-direction, the smaller is used to decomposed the x-direction
- Users may choose a preferred decomposition (nproc_x, nproc_y)
- Prime numbers and composites with large prime factors are usually to be avoided
- The behavior of 132 vs 131, and 200 vs 202 are quite different

There once was a machine next to Wal-Mart
Which before the recabling, would fall apart
    Day after day
    I always seem to say
Have this job finish ‘ere this life I depart

Samuel Coleridge
Rime of the Ancient Modeler

- How many
- How to use
- How to choose

Just How Many Cores Do I Need

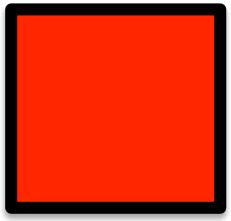
- The WRF model timing is sensitive to the selection of model options and various domain configurations.
- For a fixed grid size (km) and number of grid cells, the choice of microphysics and radiation impact the model run-time.
- For non-chemistry runs and for non-bin MP runs, the WRF model is not a memory hog.
- Horizontal domain decomposition is used.

Just How Many Cores Do I Need

- Assume three different domains:

  A

1000x1000

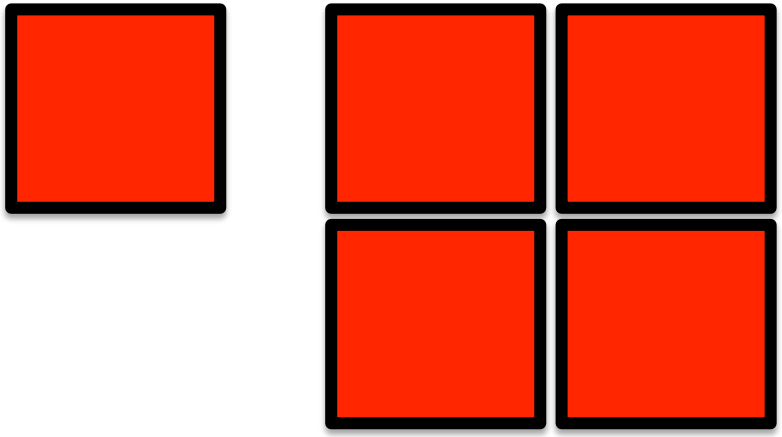


Just How Many Cores Do I Need

- Assume three different domains:

A B

1000x1000 2000x2000

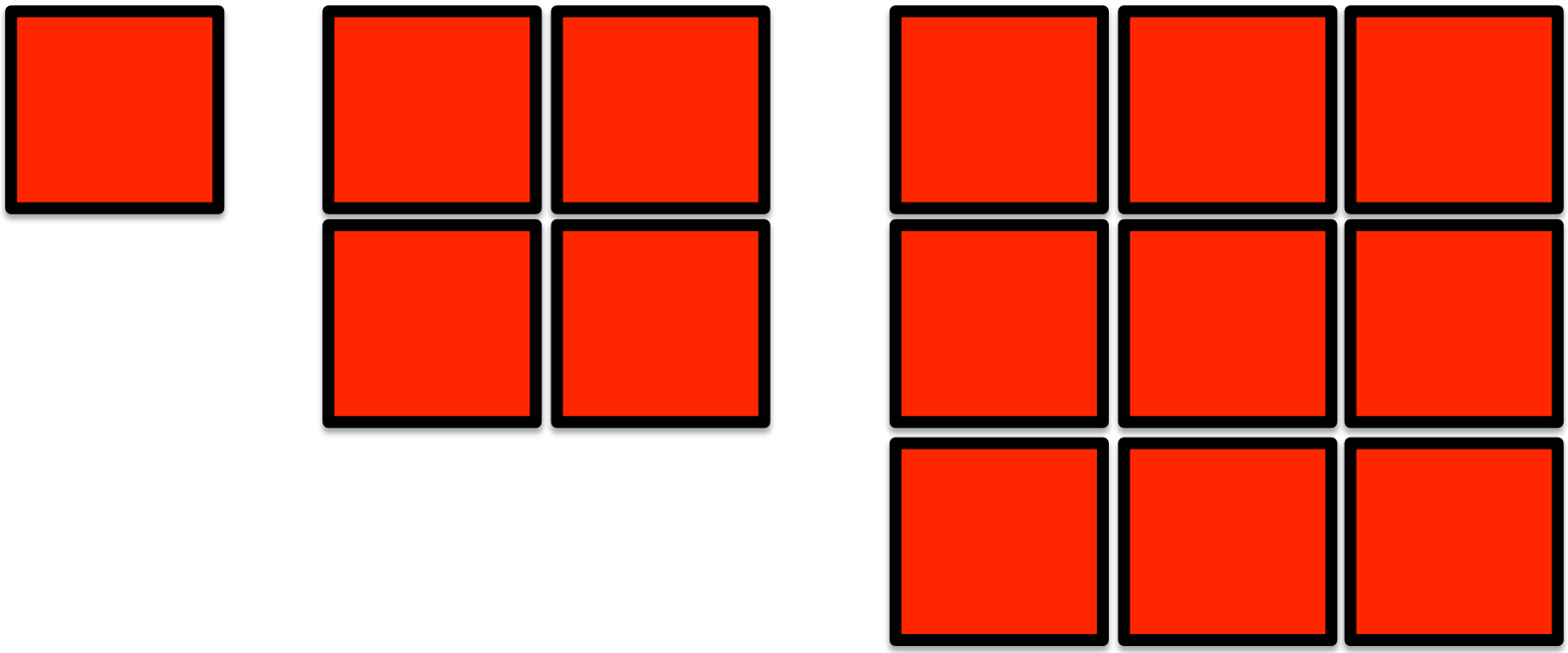


Just How Many Cores Do I Need

- Assume three different domains:

A B C

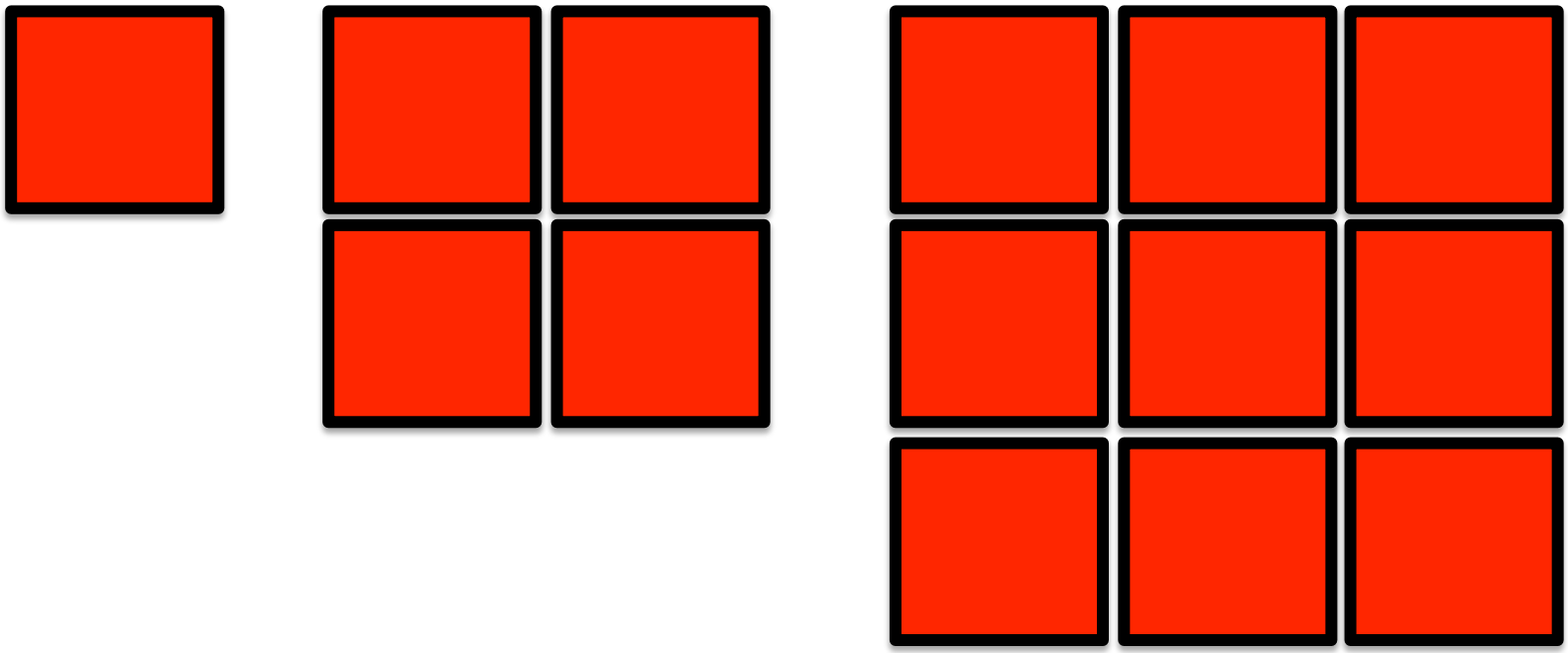
1000x1000 2000x2000 3000x3000



Just How Many Cores Do I Need

- If domain A fits on a ***n*** cores, then domains B and C fit on 4***n*** and 9***n*** cores, respectively

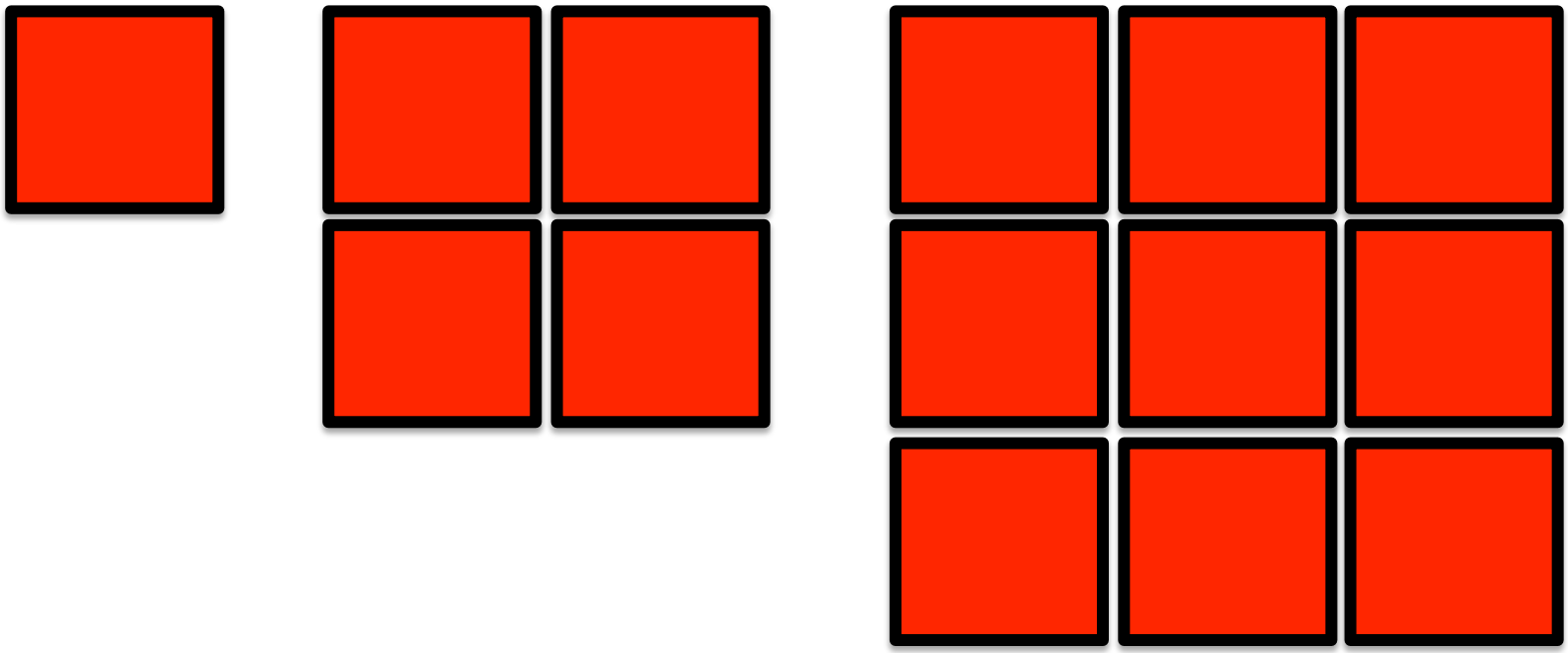
A B C



Just How Many Cores Do I Need

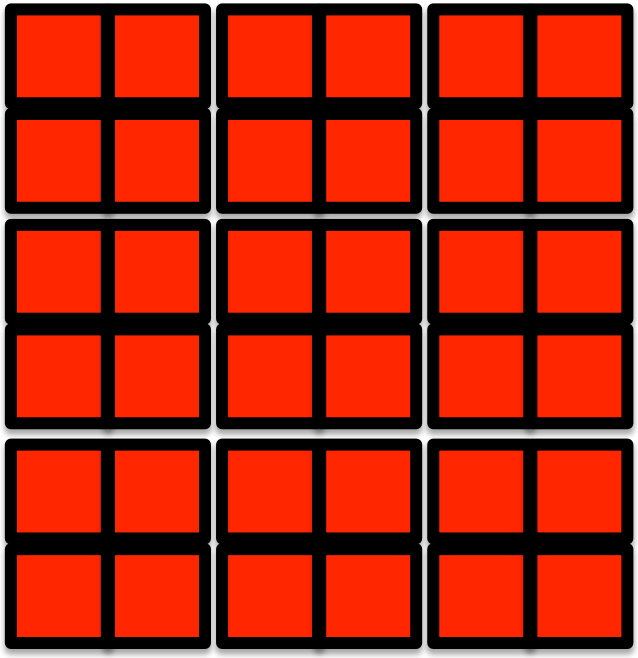
- The amount of wall-clock time for domain B (using 4***n*** cores) ~= domain C (using 9***n*** cores)

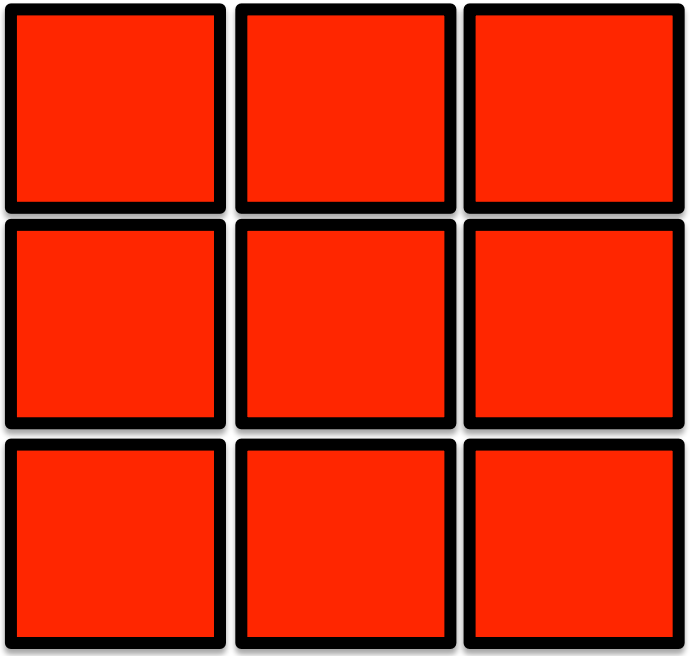
A B C



Just How Many Cores Do I Need

- The larger the decomposed sub-domain, the more efficiently it uses the core (more work with similar-ish amounts of communication).





Just How Many Cores Do I Need

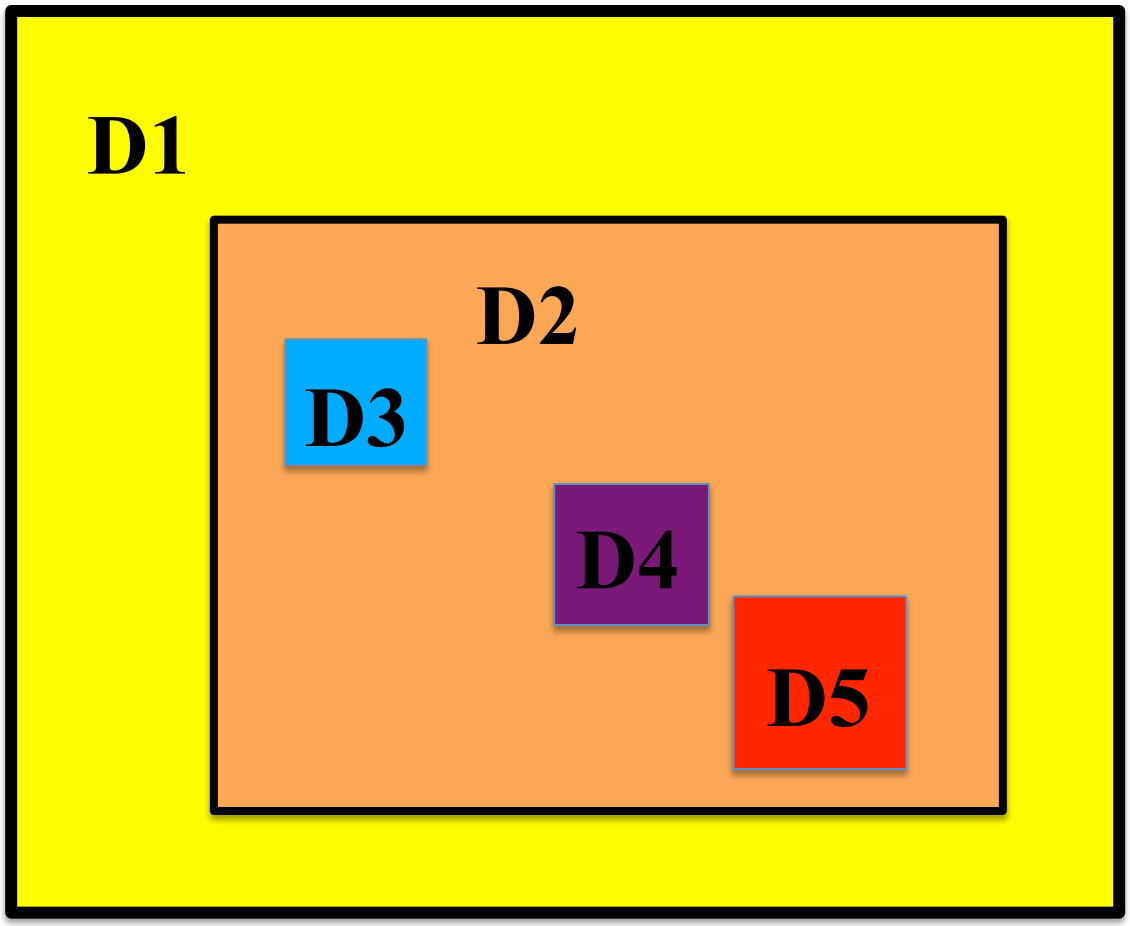
- Scaling a WRF job is straightforward.
- If a 3000x2000 domain is the desired domain size of the eventual WRF simulation …
  - 600 (600=30x20) 100x100 sub-domains could be manufactured
  - Short timings on a 200x200 domain (four sub-domains of 100x100) would yield similar performance characteristics

Just How Many Cores Do I Need

- WRF default decompositions take the two closest factors.
  - 144 total cores is a 12x12 core set up, not 72x2
- Be careful with core counts with accidentally large prime factors.
  - 128 cores using 6 I/O processors gives 122 total computational cores, decomposing as 61x2
- Decomposed domains should be larger than 10 cells on a side, and ***larger still for performance***.

Nesting

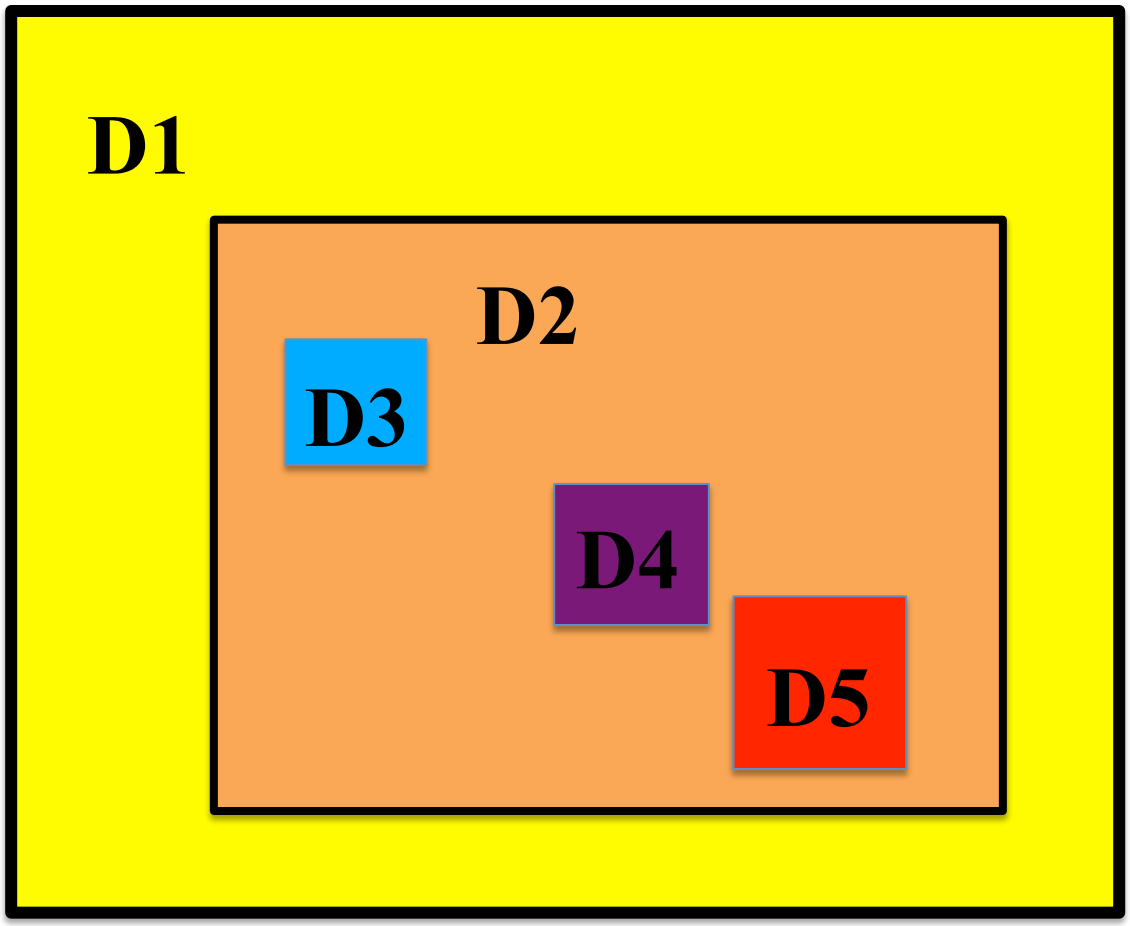
WRF 5-domain run: Domain 1 (a single 3 min dt), then Domain 2 (a single 1 min dt). Then Domain 3, in 20 s pieces up to 1 min. Then Domain 4, in 20 s pieces up to 1 min, and same with Domain 5.



WRF 5-domain run: Domain 1 (a single 3 min dt), then Domain 2 (a single 1 min dt). Then Domain 3, in 20 s pieces up to 1 min. Then Domain 4, in 20 s pieces up to 1 min, and same with Domain 5.

The order of integration is

```
D1

D2          D2

D3  D3  D3

D4  D4  D4

D5  D5  D5
```



WRF 5-domain run: Domain 1 (a single 3 min dt), then Domain 2 (a single 1 min dt). Then Domain 3, in 20 s pieces up to 1 min. Then Domain 4, in 20 s pieces up to 1 min, and same with Domain 5.

The order of integration is

```
D1

D2          D2

D3  D3  D3

D4  D4  D4

D5  D5  D5
```

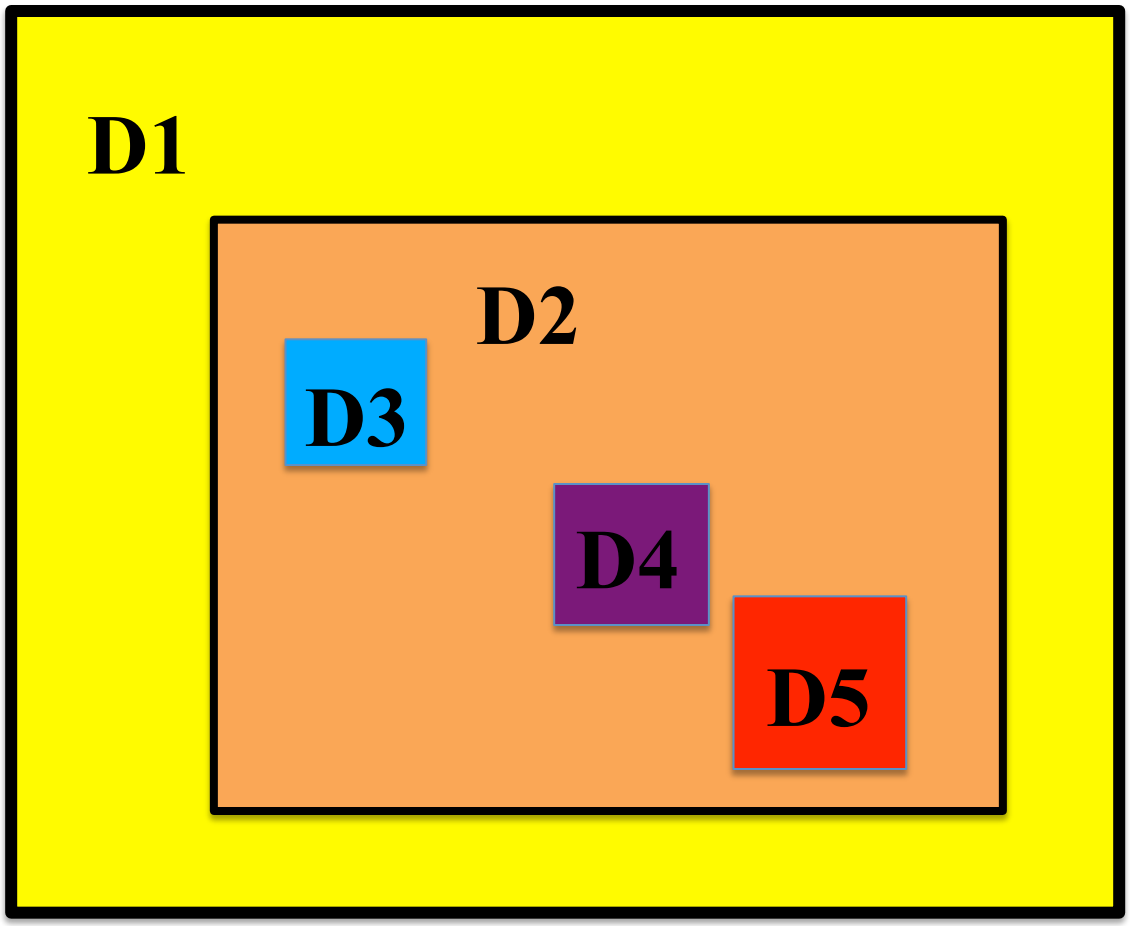
1

2

3,4,5

6,7,8

9,10,11



WRF 5-domain run: Domain 1 (a single 3 min dt), then Domain 2 (a single 1 min dt). Then Domain 3, in 20 s pieces up to 1 min. Then Domain 4, in 20 s pieces up to 1 min, and same with Domain 5.

The order of integration is

```
D1
D2          D2

D3  D3  D3

D4  D4  D4

D5  D5  D5
```

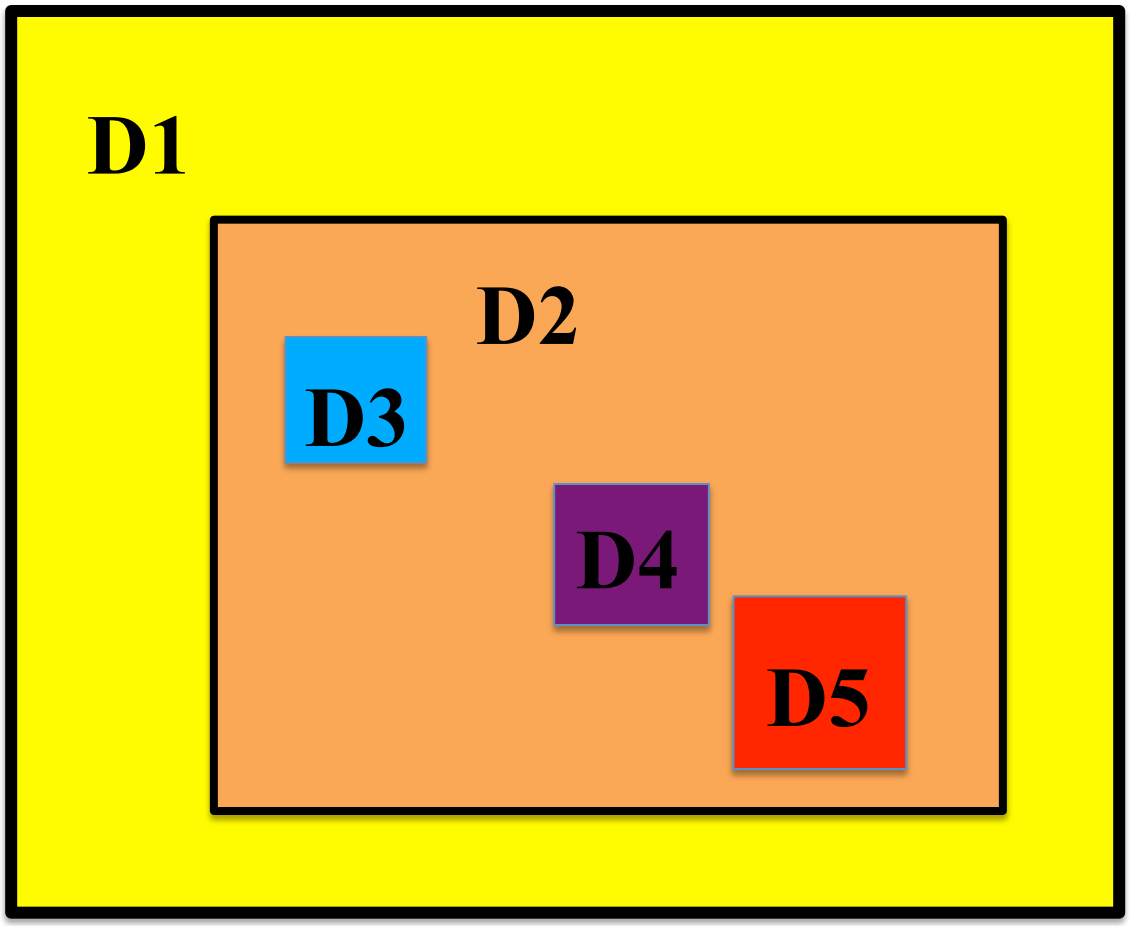
1

2

3,4,5

6,7,8

9,10,11



WRF 5-domain run: Domain 1 (a single 3 min dt), then Domain 2 (a single 1 min dt). Then Domain 3, in 20 s pieces up to 1 min. Then Domain 4, in 20 s pieces up to 1 min, and same with Domain 5.

The order of integration is

D1

D2

D3 D3 D3

D4 D4 D4

D5 D5 D5

D2

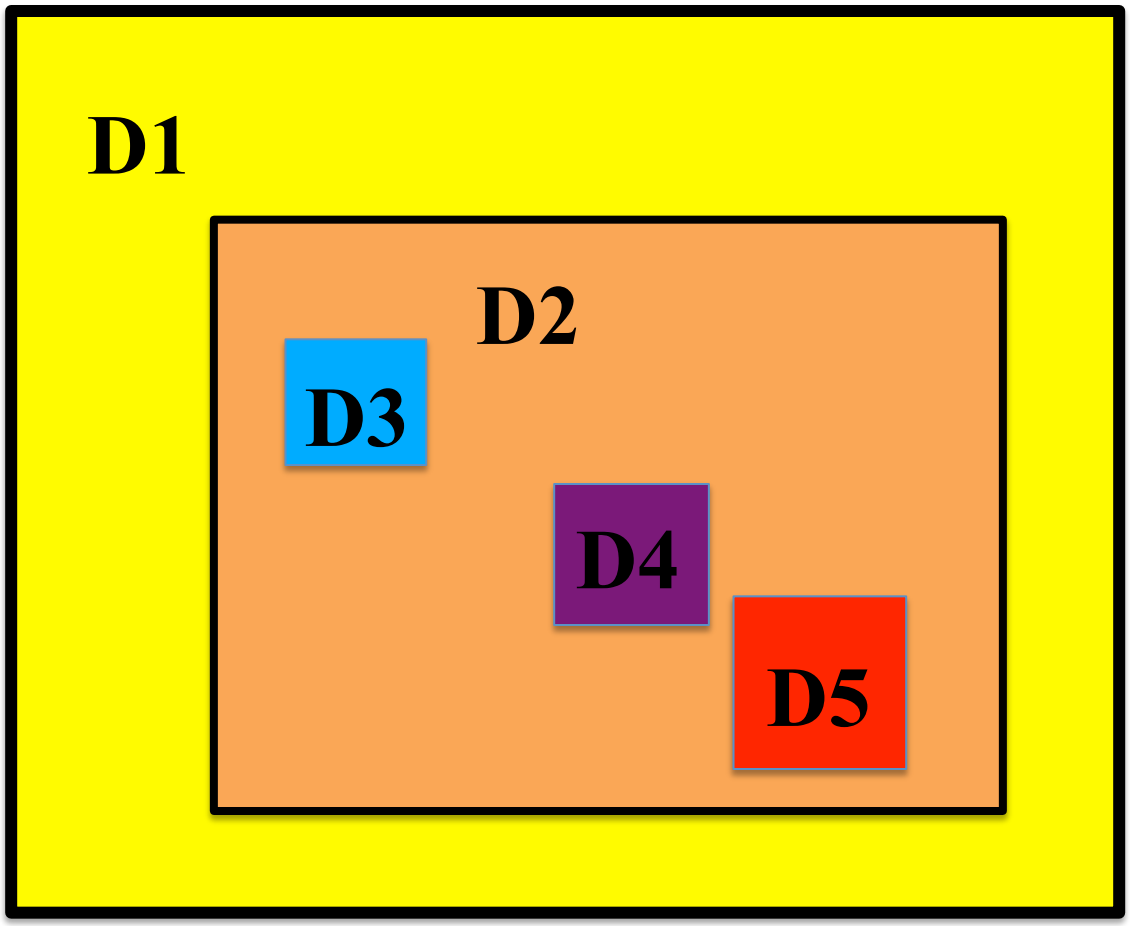
1

2

3,4,5

6,7,8

9,10,11



WRF 5-domain run: Domain 1 (a single 3 min dt), then Domain 2 (a single 1 min dt). Then Domain 3, in 20 s pieces up to 1 min. Then Domain 4, in 20 s pieces up to 1 min, and same with Domain 5.

The order of integration is

D1

D2 D2

D3 D3 D3

D4 D4 D4

D5 D5 D5

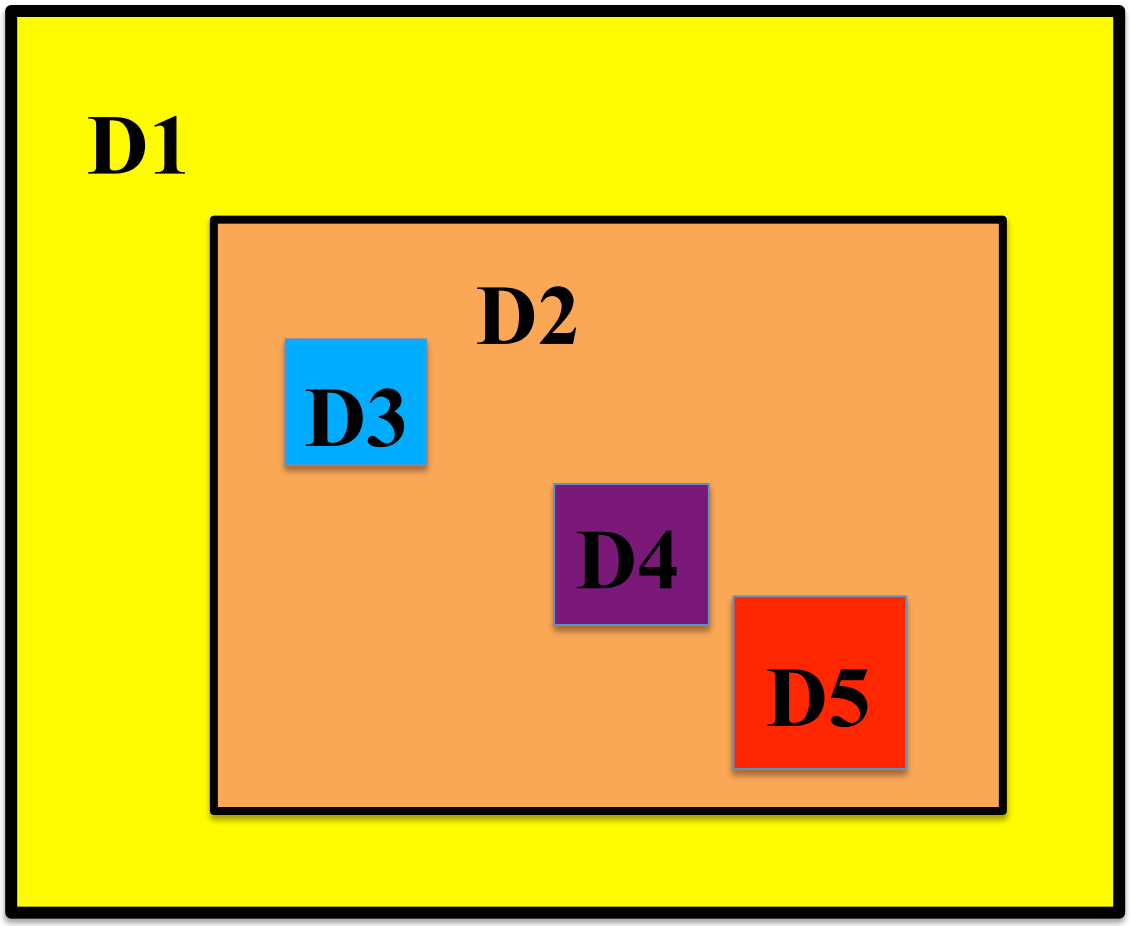
1

2

3,4,5

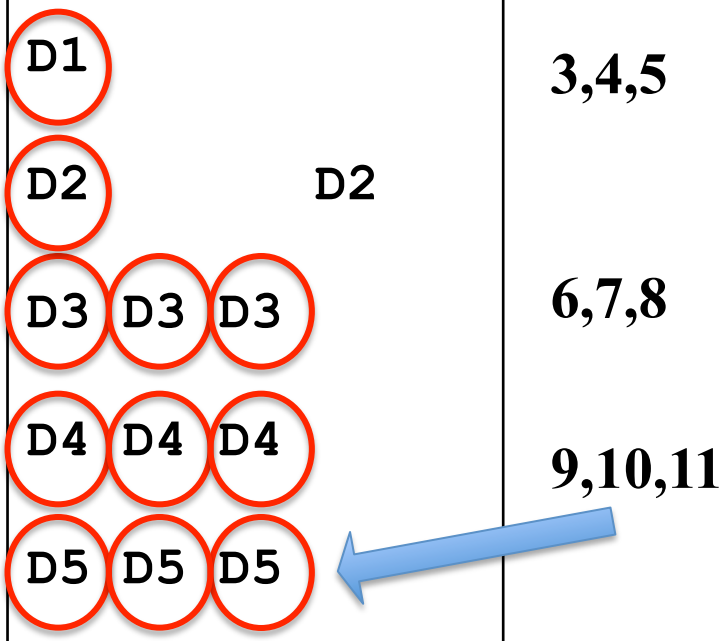
6,7,8

9,10,11



WRF 5-domain run: Domain 1 (a single 3 min dt), then Domain 2 (a single 1 min dt). Then Domain 3, in 20 s pieces up to 1 min. Then Domain 4, in 20 s pieces up to 1 min, and same with Domain 5.

The order of integration is



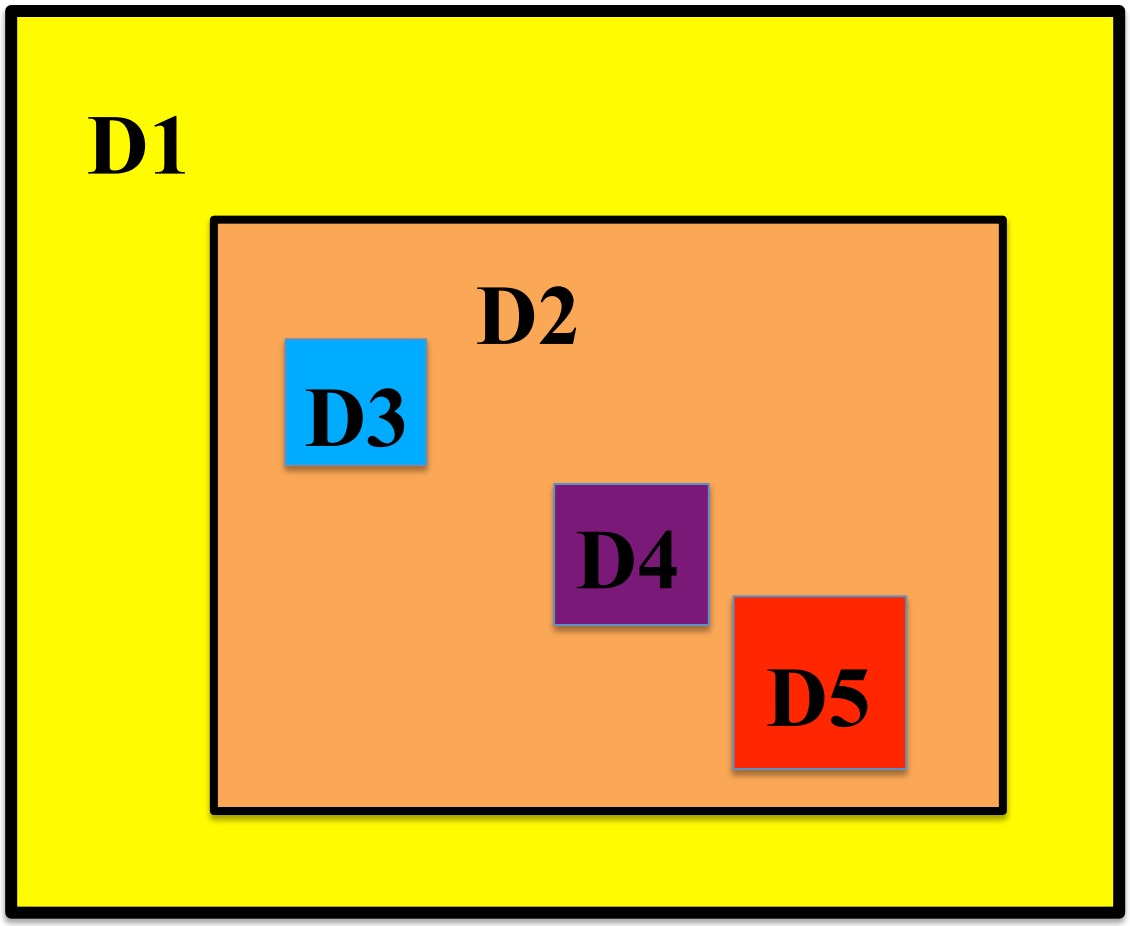
1

2

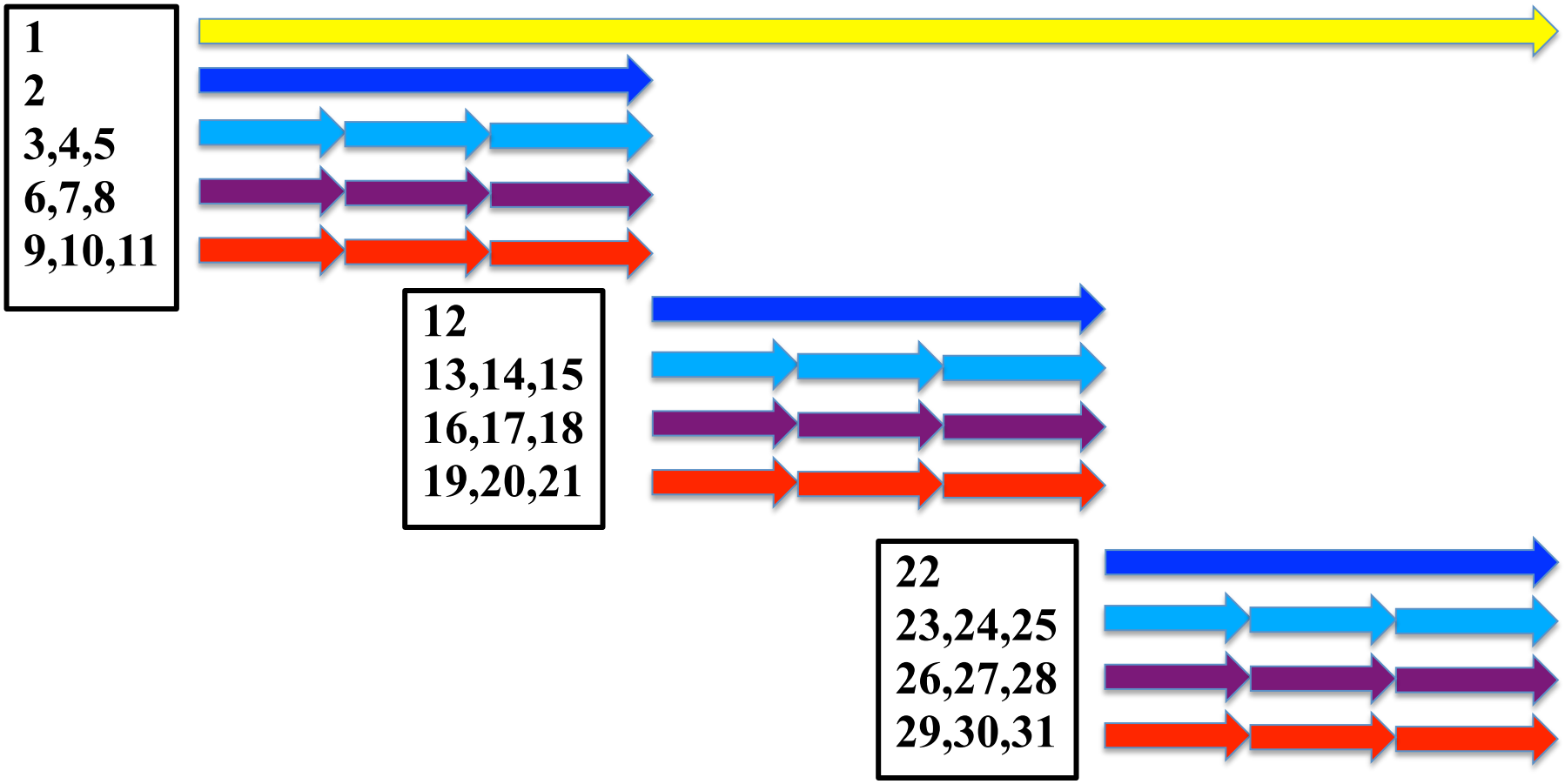
3,4,5

6,7,8

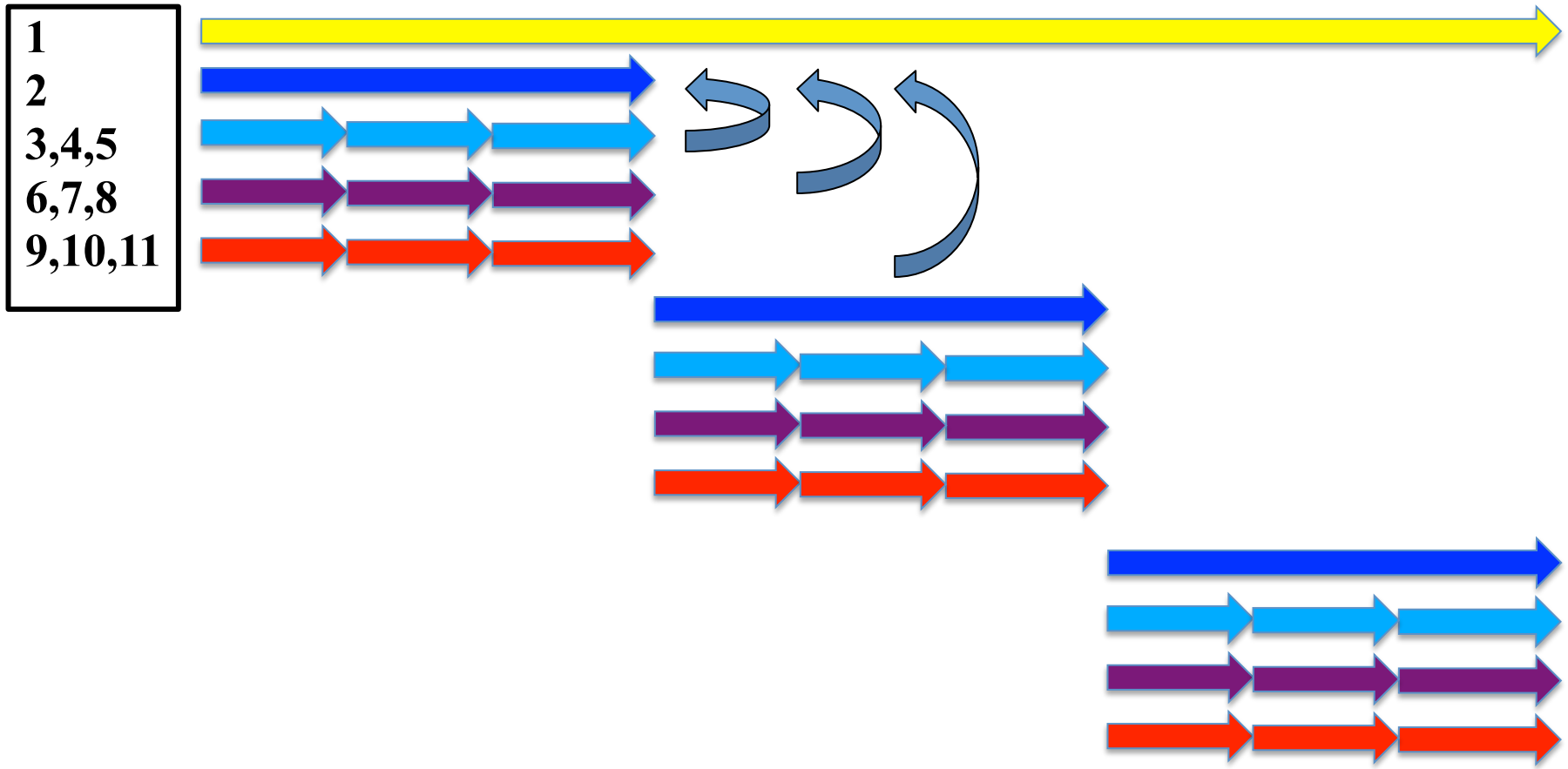
9,10,11



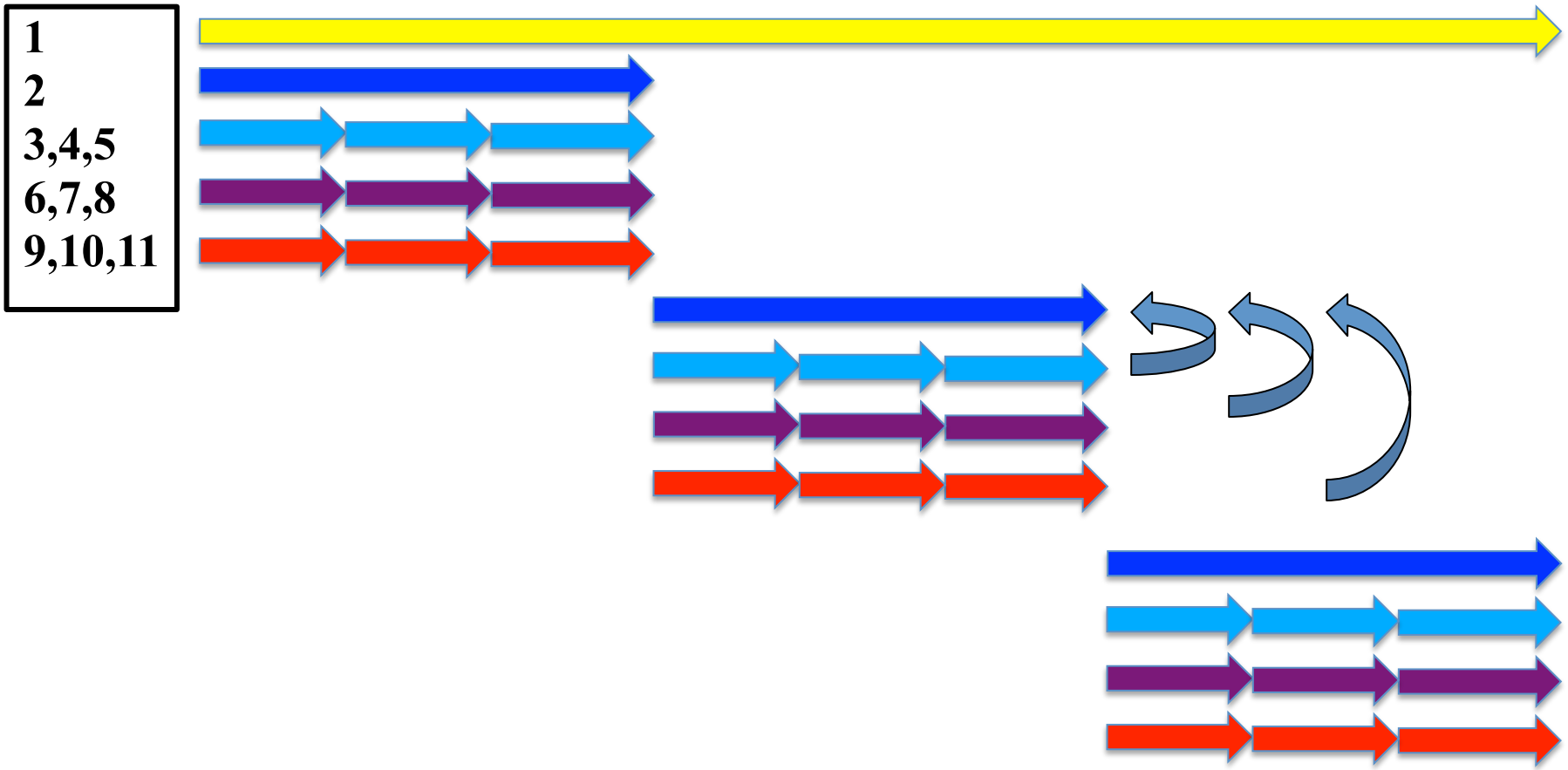
WRF 5-domain run: Domain 1 (a single 3 min dt), then Domain 2 (a single 1 min dt). Then Domain 3, in 20 s pieces up to 1 min. Then Domain 4, in 20 s pieces up to 1 min, and same with Domain 5.



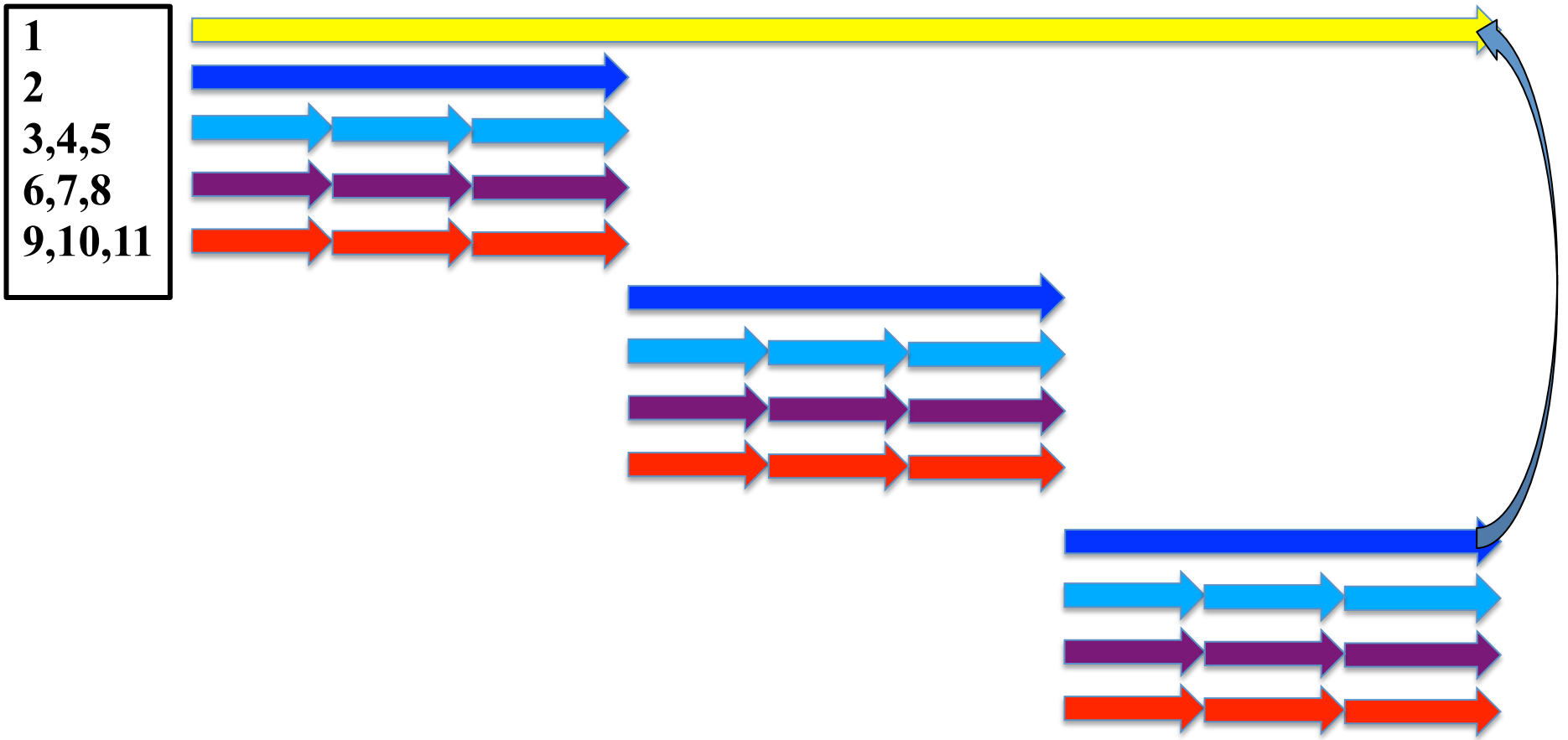
WRF 5-domain run: Domain 1 (a single 3 min dt), then Domain 2 (a single 1 min dt). Then Domain 3, in 20 s pieces up to 1 min. Then Domain 4, in 20 s pieces up to 1 min, and same with Domain 5.



WRF 5-domain run: Domain 1 (a single 3 min dt), then Domain 2 (a single 1 min dt). Then Domain 3, in 20 s pieces up to 1 min. Then Domain 4, in 20 s pieces up to 1 min, and same with Domain 5.



WRF 5-domain run: Domain 1 (a single 3 min dt), then Domain 2 (a single 1 min dt). Then Domain 3, in 20 s pieces up to 1 min. Then Domain 4, in 20 s pieces up to 1 min, and same with Domain 5.



Nesting Suggestions – CG Size

- The size of the nested domain may need to be chosen with computing performance in mind.
- Assuming a 3:1 ratio and the same number of grid cells in the parent and nest domains, the fine grid will require 3x as many time steps to keep pace with the coarse domain.
- A simple nested domain forecast is approximately 4x the cost of just the coarse domain.
- Don't be *cheap* on the coarse grid, doubling the CG points results in only a 25% nested forecast time increase.

Nesting Suggestions – Cost

- Example: assume 3:1 nest ratio

If the nest has the same number of grid cells, then the **amount of CPU** to do a single time step for a coarse grid (CG) and a fine grid step (FG) is **approximately the same**.

Since the fine grid (3:1 ratio) has 1/3 the grid distance, it requires 1/3 the model time step. Therefore, the FG requires 3x the CPU to catch up with the CG domain.

Nesting Suggestions – Same Area

- Example: assume 3:1 nest ratio

If you try to cover the SAME area with a FG domain as a CG domain, you need **(ratio)^2 grid points**.

With the associated FG time step ratio, you require a **(ratio)^3**.

With a 3:1 ratio, a FG domain covering the same area as a CG domain **requires 27x CPU**.

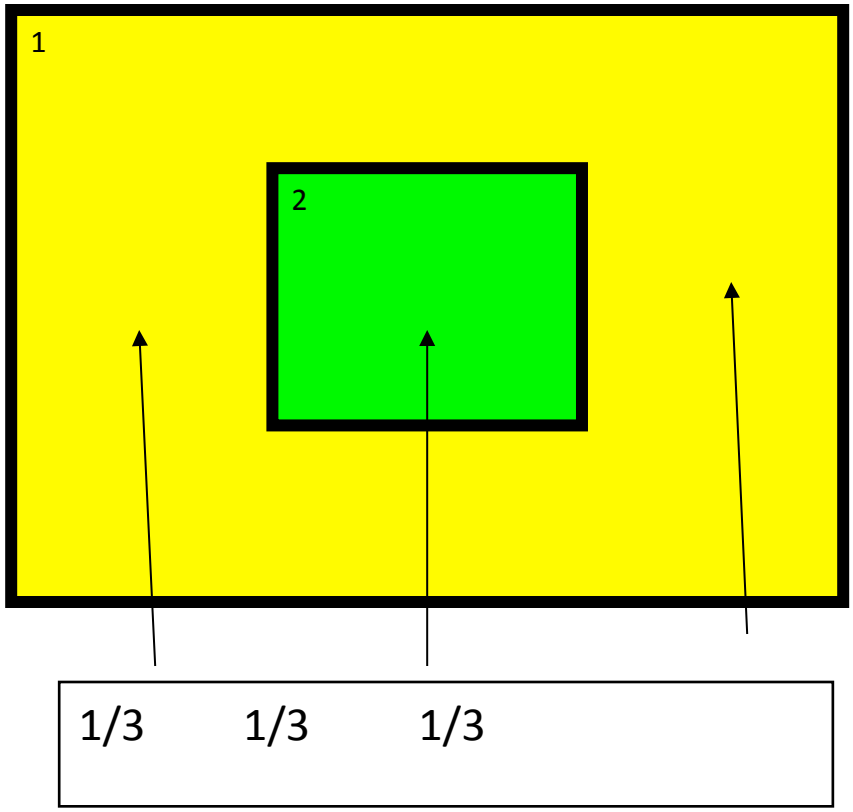
Nesting Suggestions – Same Area

- Example: assume **10:1 nest ratio**

To change your test case from 50-km resolution to a finer 5-km resolution would be at least **1000x more** expensive.

Nesting Suggestions - Location

- The minimum distance between the nest boundary and the parent boundary is FOUR grid cells
- You should have a **MUCH** larger buffer zone
- It is not unreasonable to have approximately 1/3 of your coarse-grid domain surrounding each side of your nest domain



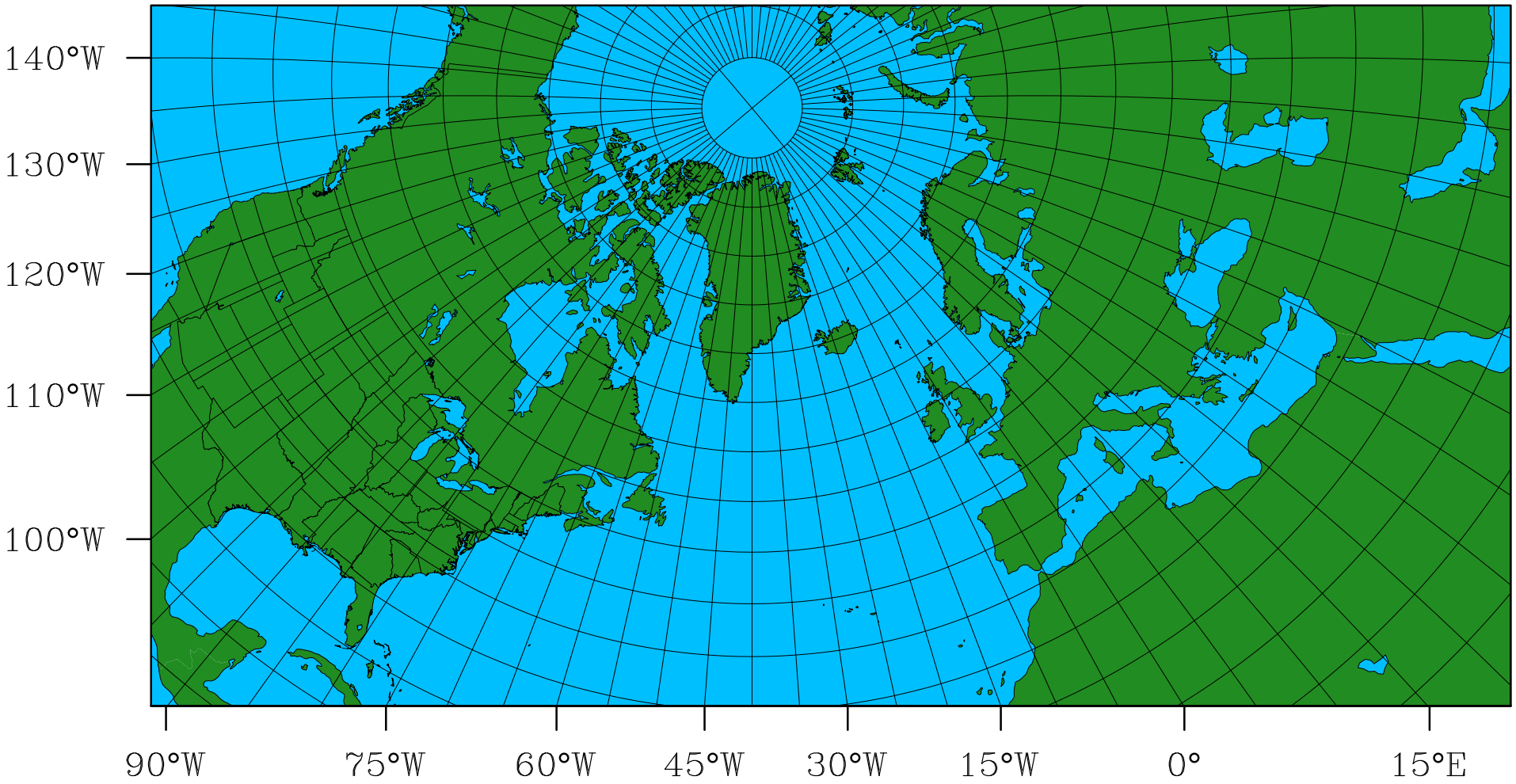
Nesting Suggestions – Inside Out

- Start with designing your inner-most domain. For a traditional forecast, you want everything important for that forecast to be entirely contained inside the domain.
- Then start adding parent domains at a 3:1 or 5:1 ratio. A parent should not have a smaller size (in grid points). Keep adding domains until the most coarse WRF grid has no more than a 3:1 to 5:1 ratio to the external model (first guess) data.

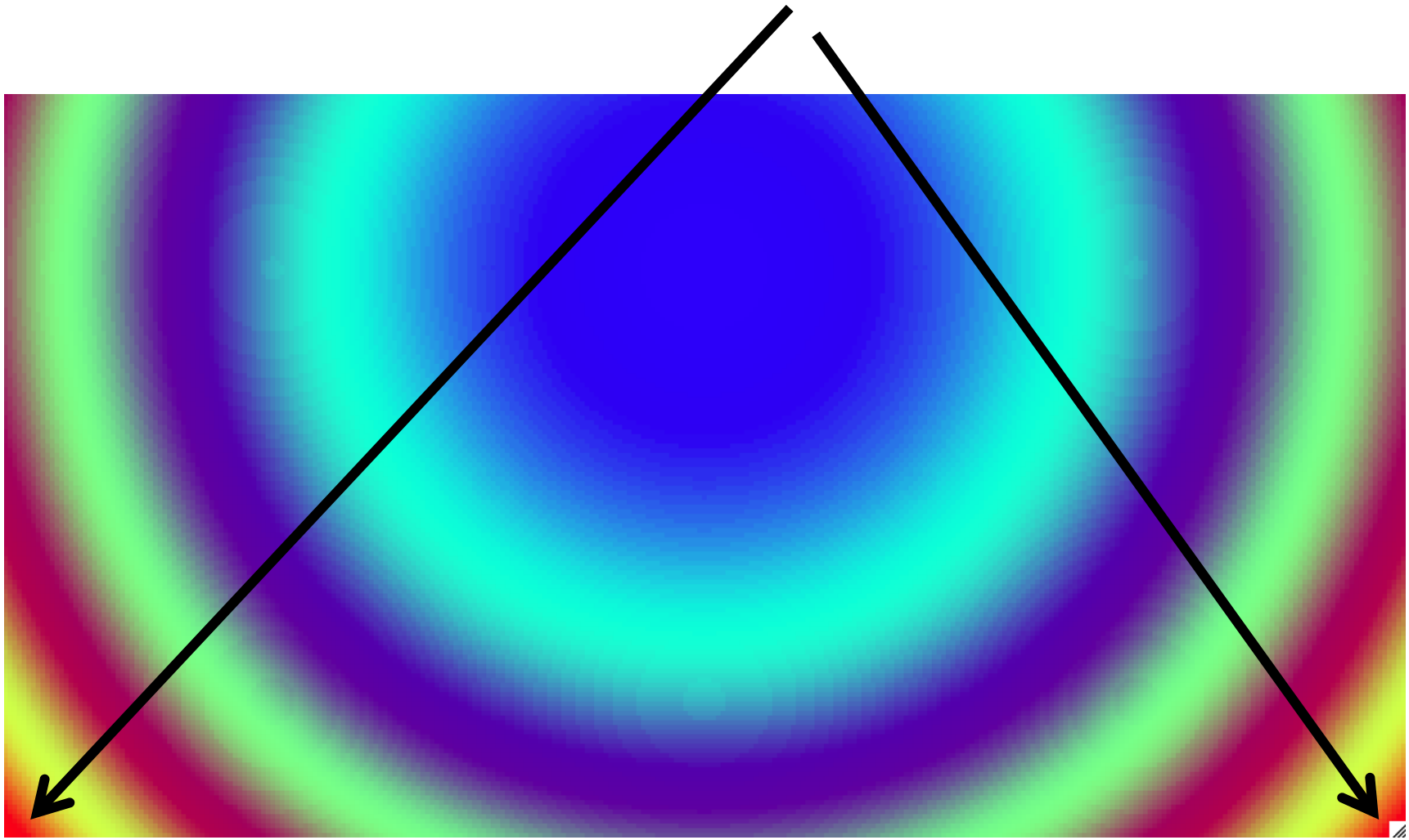
Nesting Suggestions – Big CG

- Larger domains tend to be better than smaller domains.
- A 60 m/s parcel moves at > 200 km/h. A 2-km resolution grid with 100x100 grid points could have all of the upper-level initial data swept out of the domain within a couple of hours.

Nesting Suggestions – CG dt



Map factors > 1.6



Nesting Suggestions – CG dt

- The most-coarse domain may have a geographic extent that causes large map factors.

```
time_step                = 300 (BLOWS UP)
dx                       = 50000,16666,5555
grid_id                  = 1,    ,2    ,3
parent_id                = 0,    ,1    ,2
parent_grid_ratio        = 1,    ,3    ,3
parent_time_step_ratio   = 1,    ,3    ,3
```

Nesting Suggestions – CG dt

- Reducing the time step so that the coarse grid is stable makes the model too expensive. 1.5x more

```
time_step                = 200 (STABLE, PRICEY)
dx                       = 50000,16666,5555
grid_id                  = 1,    ,2    ,3
parent_id                = 0,    ,1    ,2
parent_grid_ratio        = 1,    ,3    ,3
parent_time_step_ratio   = 1,    ,3    ,3
```

Nesting Suggestions – CG dt

- Only reduce the time step on the coarse grid, and keep the fine grid time steps at their approx original values.

```
time_step                = 200 (STABLE, CHEAP)
dx                       = 50000,16666,5555
grid_id                  = 1,    ,2    ,3
parent_id                = 0,    ,1    ,2
parent_grid_ratio        = 1,    ,3    ,3
parent_time_step_ratio   = 1,    ,2    ,3
```

Nesting Suggestions – CG dt

Domain Number	Original Time Step (s) UNSTABLE	Safe Time Step (s) STABLE EXPENSIVE	BETTER Time Step (s) STABLE CHEAPER
Domain 01 PARENT	300	200	200
Domain 02 CHILD	100	66.6	100

```
time_step                  = 300 (UNSTABLE)
parent_time_step_ratio     = 1,    ,3     ,3
```

Nesting Suggestions – CG dt

Domain Number	Original Time Step (s) UNSTABLE	Safe Time Step (s) STABLE EXPENSIVE	BETTER Time Step (s) STABLE CHEAPER
Domain 01 PARENT	300	200	200
Domain 02 CHILD	100	66.6	100

```
time_step                  = 200 (STABLE, PRICEY)
parent_time_step_ratio     = 1,    ,3     ,3
```

Nesting Suggestions – CG dt

Domain Number	Original Time Step (s) UNSTABLE	Safe Time Step (s) STABLE EXPENSIVE	BETTER Time Step (s) STABLE CHEAPER
Domain 01 PARENT	300	200	200
Domain 02 CHILD	100	66.6	100

```
time_step                  = 200 (STABLE, CHEAP)
parent_time_step_ratio     = 1,    ,2     ,3
```

Nesting Suggestions – CG dt

- Model time step is always proportional to the time step of the most coarse grid.
- The coarse grid is the only grid impacted with large map factors: **`dt(s) = 6*dx(km)`**
- The nominal grid distance always needs to be scaled:

  dt(s) = 6*dx(km) / **MAX (map factor in domain)**
- Reducing the coarse grid time step does not significantly reduce model performance if you can tweak the time step ratio.

Nesting Suggestions – CG dt

- **The take away:**

- The time step ratio and grid distance ratio are not necessarily identical, and may used effectively when large map factors in the coarse grid domain force a time step reduction for stability.
- If map factors are causing stability troubles, it is usually only the most coarse grid that is impacted since the fine grid is usually in the middle of the domain.

Nesting Suggestions - Wrap Up

- Set up domain first to provide good valid forecast, then deal with efficiency
- Selecting a set of domains with the reason "it is all I can afford" gets you into trouble
- Numerically stable and computationally expedient do not imply scientifically or physically valid

Talk #2: WRF Optimization Best Practices

- I/O (though mostly "O")
  - netcdf3 vs netcdf4 (HDF5 compression)
  - PNETCDF
  - Quilting
  - io_form = 102
  - Run-time auxiliary streams
  - Vertical interpolation
  - Diagnostics
- Benchmarking
  - How to
  - What to avoid
  - Available benchmarks
  - Using and interpreting benchmark information
- Debugging
  - real*4 vs real*8
  - ./configure –d and -D

IO

WRF I/O

- By default WRF has netcdf (classic, small file format), binary (internal one-off format), and grib1 formats available. For portability, netcdf (io_form=2) is the favored default.
- At compile time, users may request PNETCDF
  - $PNETCDF points to bin lib dir for PNETCDF
  - io_form = 11
- Typically, PNETCDF is used in combination with WRF QUILTING.
- nio_tasks_per_group = 0 => How many cores are used per group for IO. Zero deactivates the option, "2" uses two MPI tasks per group for IO
- nio_groups = 1 => usually "one" group per stream (3 domains, each with history and a restart: 6 nio_groups)

WRF I/O

- Streams (similar to Fortran units): pathways into and out of model
- Can be thought of as files, though that is a restriction
  - History + auxiliary output streams (10 and 11 are reserved for nudging)
  - Input + auxiliary input streams (10 and 11 are reserved for nudging)
  - Restart, boundary, and a special DA in-out stream
  - Currently, 24 total streams
  - Use the large values and work down to stay away from "used"
  - Non-chemistry: use history streams 13-22, 24
  - Chemistry: use history streams 20, 21, 22, 24

WRF I/O

- Attributes of streams
  - Variable set
    - The set of WRF state variables that comprise one read or write on a stream
    - Defined for a stream at compile time in Registry
  - Format
    - The format of the data outside the program (e.g. NetCDF), split
    - Specified for a stream at run time in the namelist

WRF I/O

- Attributes of streams
  - Additional namelist-controlled attributes of streams
    - Dataset name
    - Time interval between I/O operations on stream
    - Starting, ending times for I/O (specified as intervals from start of run)

Example 1: Add output without recompiling

- **Edit the namelist.input file, the time_control namelist record**

```
iofields_filename = "myoutfields.txt"  (MAXDOM)
io_form_auxhist24 = 2  (choose an available stream)
auxhist24_interval = 10 (MAXDOM, every 10 minutes)
```

- **Place the fields that you want in the named text file `myoutfields.txt`**

```
+:h:24:RAINC,RAINNC
```

- **Where "+" means ADD this variable to the output stream, "h" is the history stream, and "24" is the stream number**

Data, data every where, nor any drop to drink

Samuel Coleridge

Rime of the Ancient Modeler

- Output data more quickly
- Output data more smallly
- Output data more lessly

Example 1: Zap output without recompiling

- **Edit the namelist.input file, the time_control namelist record**

```
iofields_filename = "myoutfields.txt"
```

- **Place the fields that you want in the named text file `myoutfields.txt`**

```
-:h:0:W,PB,P
```

- **Where "–" means REMOVE this variable from the output stream, "h" is the history stream, and "0" is the stream number (standard WRF history file)**

(De)Selecting Model Output Fields

- Several years ago John Michalakes provided a simple run-time option to add and remove fields from WRF streams

```
&time_control
iofields_filename = "myoutfields.txt"
/
```

```
-:h:0:W,PB,P
```

(De)Selecting Model Output Fields

- Particularly helpful when ncview shows:

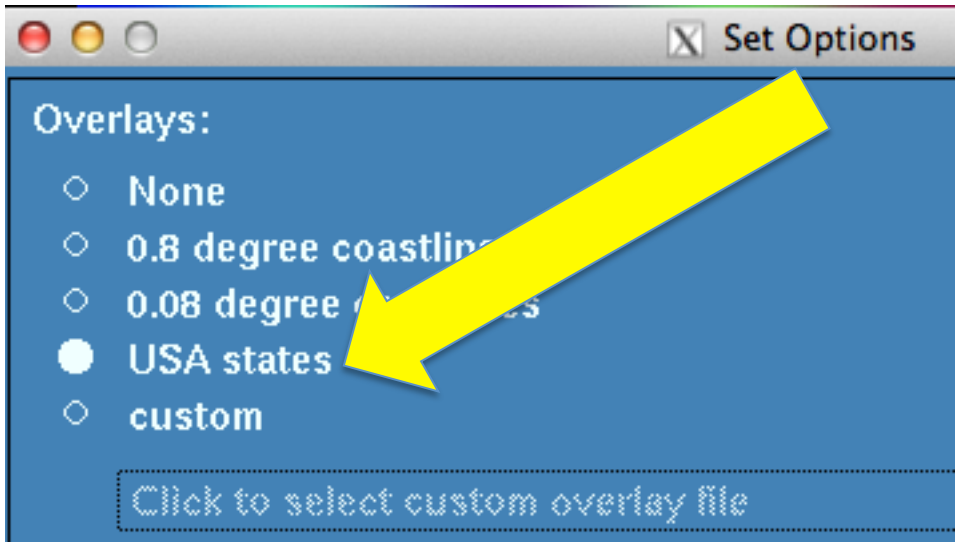
(De)Selecting Model Output Fields

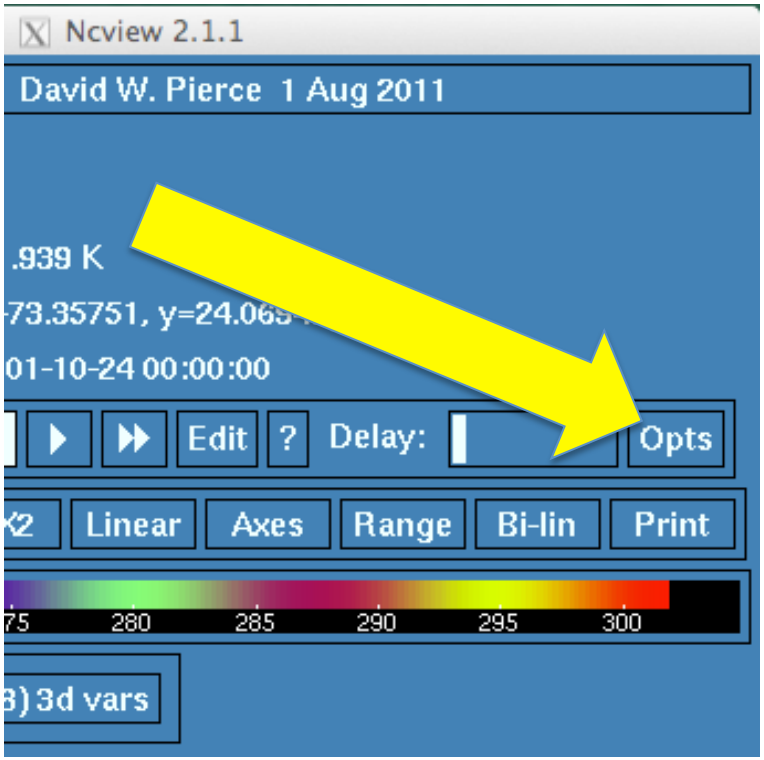
- Removing half of the unwanted or never used 3d arrays cuts your file sizes in half
- Default values for “history” that are in the Registry do not obligate users

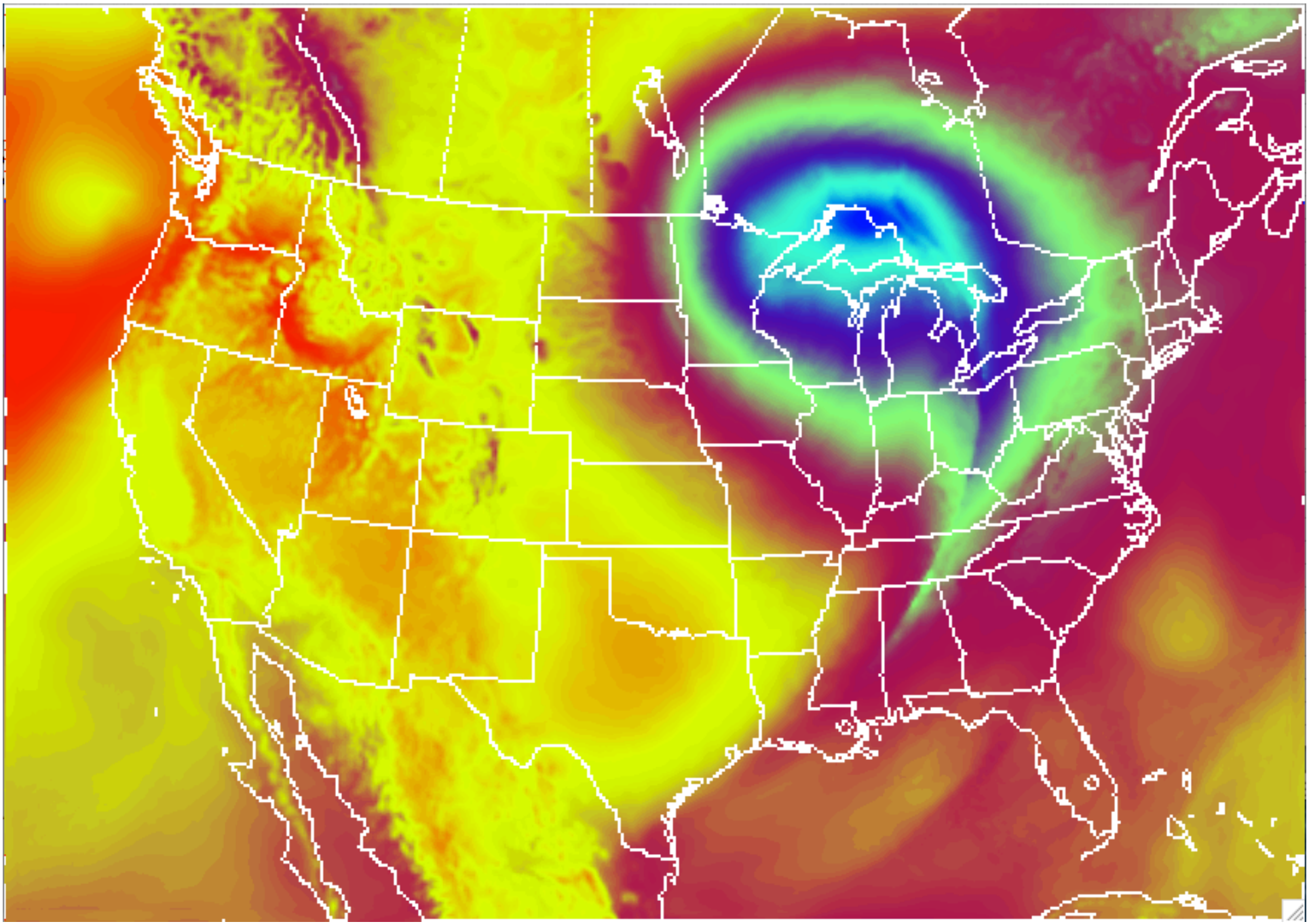
CF Compliant-ish

If you output the WRF model data with a single time period per file, ncview is able to recognize the WRF projections.

```
&time_control
frames_per_outfile = 1, 1,
/
```







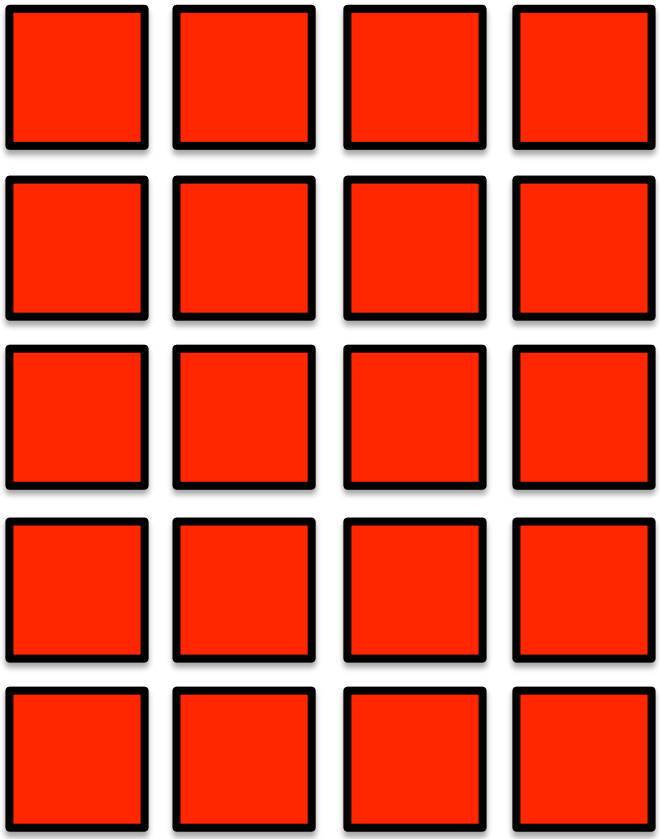
Stitching Model Output Together

- Yunheung Wang (CAPS) developed and Kevin Manning improved a scheme that joins “split data” back together

```
&time_control
history_interval_s = 150,   60,    60,
io_form_history    = 102
/
```

Stitching Model Output Together

- Running on 20 cores could produce the following WRF model decomposition and output:



Stitching Model Output Together

- With large domains, model output can dominate the total wall clock time
- With the "102" option, when running on 800 cores, there are 800 files
- Files get constructed with names such as

```
wrfout_d01_2010-06-23_15:00:00_0000
wrfout_d01_2010-06-23_15:00:00_0001
…
wrfout_d01_2010-06-23_15:00:00_0799
```

Stitching Model Output Together

- The only purpose is timing performance
- Works well with multiple domains and when restarts overlap with model output times
- The joining program is DM parallel
- For a 2000x2000x100 WSM6 domain, 2 minutes per time period with 8 cores manufactured the single file
- Scripts exist to run the joining program concurrently with WRF

Stitching Model Output Together

- Single file input:

```
Timing for processing wrfinput file
(stream 0) for domain        1:
320.15085 elapsed seconds
```

- Multiple file output:

```
Timing for Writing
wrfout_d01_2010-06-23_12:00:00 for
domain        1:    0.90883 elapsed
seconds
```

NETCDF4 Compression

- Huang Wei and Jianyu Liu have put in a simple way to get impressive NETCDF4 compression with WRF model output

- If the user has NETCDF4 libraries that have HDF5 compression included, then a single "env" variable is all that is required

NETCDF4 Compression

- Prior to running **`./configure`** …

```
setenv NETCDF4 1
export NETCDF4=1
```

NETCDF4 Compression

- This is fully supported in WRF 3.5 and beyond
- File sizes tend to be about half of the original size
- The compression works well with fields which contain similar values (such as near-zero quantities for many of the hydrometeor fields)
- YS NETCDF tools support this compression: ncview, ncl, nco

NETCDF4 Compression

- It takes time to “compress” data.
- 500x500, 256 cores time to write model output (s)

	NETCDF3	NETCDF4
IO_FORM = 2	25	45
IO_FORM = 102	0.35	0.65

- Using io_form=102, the extra time is

Diagnostics

```
&time_control
 io_form_history          = 0
 io_form_auxhist23        =  2,
 auxhist23_interval       = 60,    30,     10,
 frames_per_auxhist23     =  1,     1,      1,
 auxhist23_outname        = "PLEVS_d<domain>_<date>”
/

 &diags
 p_lev_diags              = 1
 num_press_levels         = 2
 press_levels             = 50000, 25000
/
```

Diagnostics

```
&time_control
 io_form_history          = 0
 iofields_filename        = "field_list_d01.txt", …
 io_form_auxhist24        =  2
 auxhist24_interval       = 60,   30,    10,
 frames_per_auxhist24     =  1,    1,     1,
 auxhist24_outname        = "SFC_d<domain>_<date>"
/

&afwa
 afwa_diag_opt            = 1,   1,   1,
/
```

File:

+:h:24:MU,RAINC,RAINNC,U10,V10,T2,Q2,XLAT,XLONG,AFWA_MSLP, REFL_10CM

Benchmarking

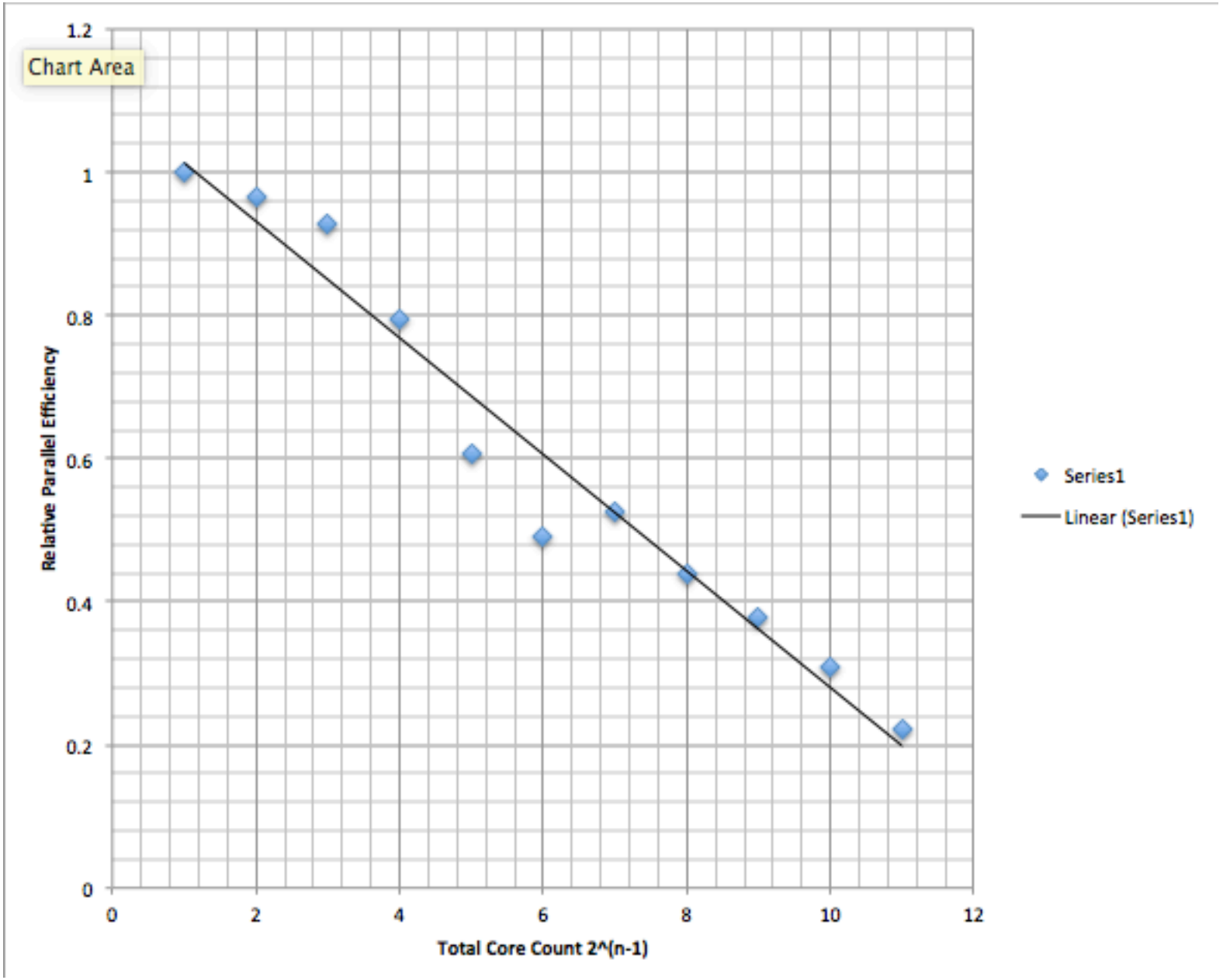


Chart Area
1.2
1
0.8
0.6
0.4
0.2
0
Relative Parallel Efficiency
0
2
4
6
8
10
12
Total Core Count 2^(n-1)
Series1
Linear (Series1)

Benchmarking

Revamping the existing benchmark files

Small one is 425x300, 12-km, 3-h: so suitable for most desktop -> department-sized systems

Will provide restart, lateral boundary, and namelist.input for all WRF releases: from **3.0.1.1** upto **3.7**

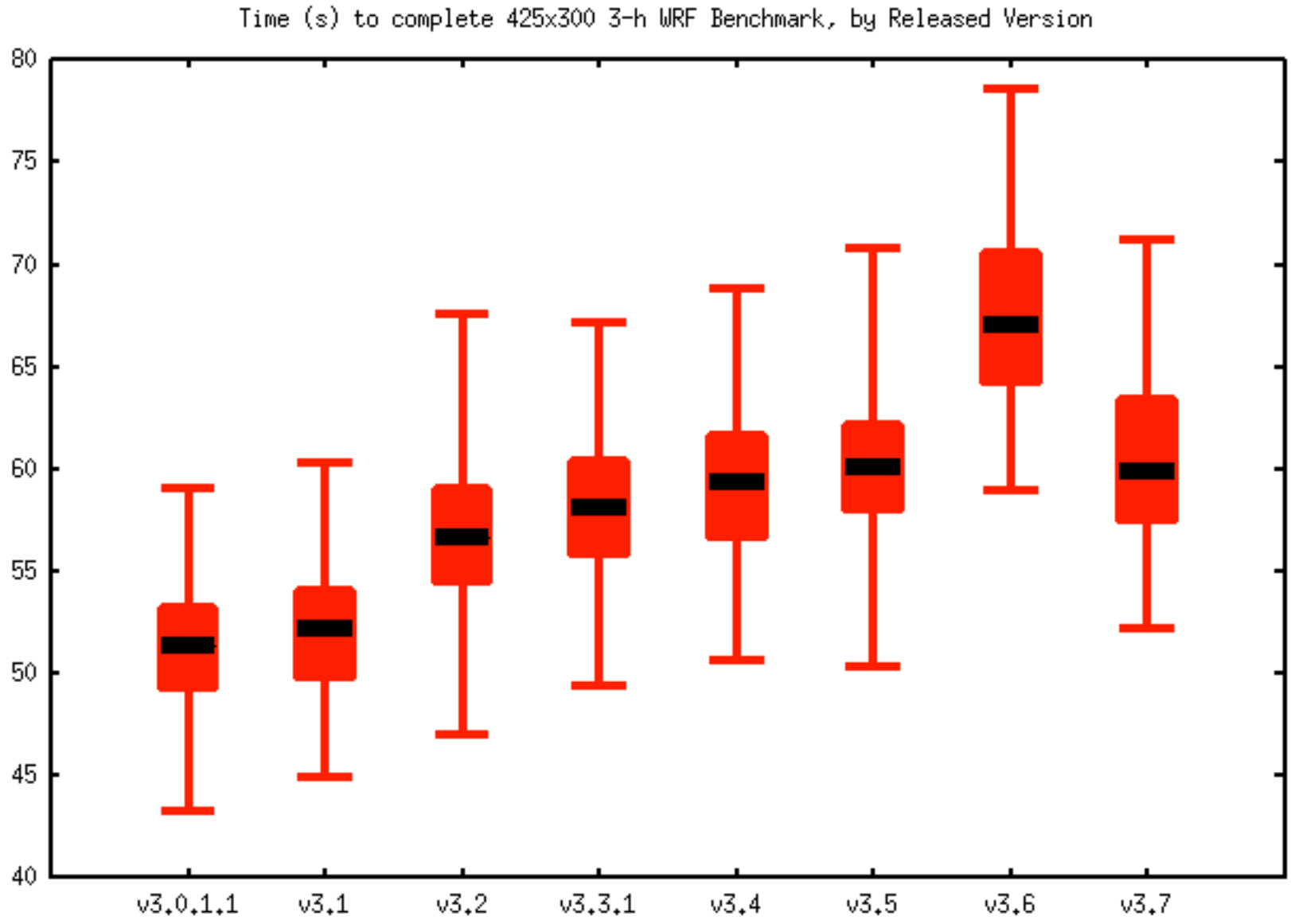
Benchmarking

- 2001 Oct 24 0000 UTC init + 24-h simulation, drop a restart
- 3-h forecast (72 s time step) gives 150 time steps
- No I/O counted in timing
- For 1 through 1024 cores (by powers of 2)

Benchmarking

- Finding **W I D E** variation in timings, but a definite slowing trend
- Ran 200 instantiations of each of 8 WRF model releases

Benchmarking



Debugging

Debugging

- What are simple recommendations for trying to debug a problem in WRF?

- The WRF build system allows the user to configure the model to run with many compiler-supplied error trapping systems activated.
- **`./clean -a`**
- **`./configure –D`**
- This executable will run VERY slowly.

Debugging

- What are simple recommendations for trying to debug a problem in WRF?

- Try to track down problems in big domains by simplifying:
- smaller domains
- single processor
- remove physics options sequentially
- short forecasts through use of restart

Debugging

- When does NCAR want to be contacted?

- When a standard model set up fails, we want to know.
- “Standard” is a recent release running with reasonable settings, and typical input data that we frequently run.

Debugging

- When does NCAR NOT want to be contacted?

- A developer wrote code, and now the WRF model fails when the option is turned on.
- A developer wrote code, and now the WRF model fails even when the option is turned off.
- A developer wrote code, and there is just no way the model failure could be because of the ABSOLUTELY SOLID code written by the developer.

Debugging

- What are typical failure modes for WRF?

- Bad initial conditions
- The model simulation fails quickly (first few time steps).
- If the first time step is OK, look for DRAMATICALLY bad fields, such as from a flag value, not an actual physically meaningful value.

Debugging

- What are typical failure modes for WRF?

- Bad initial conditions
- Too many, too few vertical levels.
- Poorly distributed vertical levels (let the real program figure them out for one of your tests).

Debugging

- What are typical failure modes for WRF?

- Bad initial conditions
- If any (i,j) info is provided by the WRF model, use a visual tool (ncview) to look at that location for these fields: MU, U, V, T, PH, QVAPOR, W.
- For 3d arrays, look completely top to bottom.
- The masked fields may be problematic at the initial time: TSLB, SMOIS, SEAICE, SST.

Debugging

- What are typical failure modes for WRF?

- Model is unstable early on
- The CFL violations are reported for values > 2.
- Values that large will kill the WRF simulation.
- Early CFL problems MIGHT be alleviated with a shorter time step.
- Modify solve_em.F to force the RK and the sound loop to have only one iteration to localize the problem.

Debugging

- What are typical failure modes for WRF?

- Model is unstable early on
- Again, early on, most troubles stem from the IC.
- Regardless of the location of the failure message (cumulus, radiation, land surface), review closely the IC file.
- The problem, for an early failure, is unlikely to be due to a problem in the radiation scheme, for example.

Debugging

- What are typical failure modes for WRF?

- Model is unstable later in the simulation
- Usually, shortening the time step is not that helpful (but any port in a storm).
- Take care to notice if the reported CFL violations in the rsl files are fatal, or just “business as usual” and the model has recovered with vertical velocity damping.
- Later-in-the-simulation failures are *hard* to solve.

Debugging

- What are typical failure modes for WRF?

- Model is unstable later in the simulation
- Is the failure reproducible – on a re-run does the WRF model fail exactly the same in exactly the same place.
- Reproducible failures allow a restart file to get a short simulation to test.
- If the restarted simulation also successfully fails, then a recompiled code with error trapping activated may help out.

Debugging

- What are typical failure modes for WRF?

- Model is unstable later in the simulation
- What is physically happening? Is the sun coming up or setting? Is this a land/water boundary? Is ice an issue at the grid cell? What is the first field impacted?

Debugging

- What are tools that NCAR WRF user support uses for debugging the model when it fails?

- With netcdf output, a number of simple visual tools are available: ncview, ncl.
- If the model shows significant sensitivity to physical parameterization settings, ncdiff for a few variables might be helpful.
- Variables to particularly consider: MU, U, V, W, T, PH, QVAPOR, TSLB, SMOIS

Debugging

- What are tools that NCAR WRF user support uses for debugging the model when it fails?

- Running with different compilers (or even different versions) is sometimes helpful.
- When a failure occurs:
- Does the model work with different ICs
- Same IC source, different day
- Different physics

Debugging

- What are tools that NCAR WRF user support uses for debugging the model when it fails?

- Floating precision can sometimes be helpful.
- ./clean –a
- ./configure –r8 (works Intel, PGI, GNU)